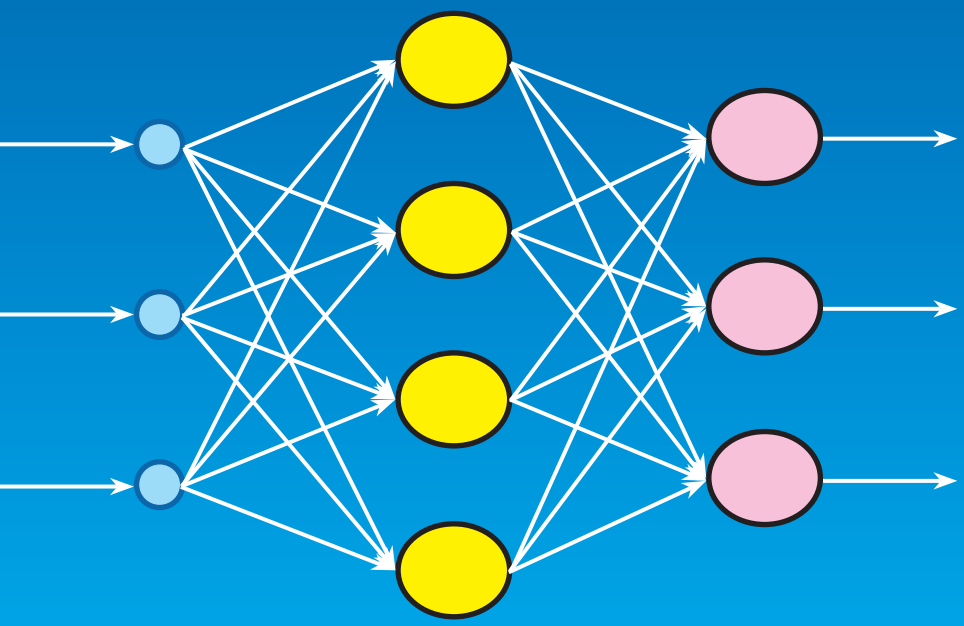
Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Институт радиоэлектроники и информационных технологий — РТФ

М. П. ТРУХИН

МОДЕЛИРОВАНИЕ СИГНАЛОВ И СИСТЕМ

Сетевые модели

Учебное пособие



Министерство науки и высшего образования
Российской Федерации
Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

М. П. Трухин

МОДЕЛИРОВАНИЕ СИГНАЛОВ И СИСТЕМ

Сетевые модели

Учебное пособие

Рекомендовано методическим советом
Уральского федерального университета
для студентов вуза, обучающихся
по направлению подготовки
11.03.01 — Радиотехника

Екатеринбург
Издательство Уральского университета
2018

УДК 621.391:004.94(075.8)
ББК 32.841-в6я73
Т80

Рецензенты:
кафедра общепрофессиональных дисциплин технических специальностей Уральского технического института связи и информатики (филиал Сибирского государственного университета телекоммуникаций и информатики) (зам. завкафедрой доц., канд. техн. наук *Н. В. Будылдина*);
проф., д-р техн. наук *В. П. Часовских* (Уральский государственный лесотехнический университет)

Научный редактор — проф., д-р техн. наук *В. Э. Иванов*

Трухин, М. П.
Т80 Моделирование сигналов и систем. Сетевые модели : учебное пособие / М. П. Трухин. — Екатеринбург : Изд-во Урал. ун-та, 2018. — 204 с.

ISBN 978-5-7996-2503-0

В пособии приведены методология и технология компьютерного моделирования систем с помощью сетевых моделей. Рассмотрены теоретические и практические вопросы приложений формализма транспортных сетей, сетей Петри и *E*-сетей, а также нейронных сетей для разработки и анализа математических моделей сложных систем, в том числе в среде MATLAB.

Учебное пособие предназначено для студентов, изучающих вопросы системного моделирования и обработки сигналов, в том числе программные и аппаратные методы защиты информации в телекоммуникационных системах.

Библиогр.: 40 назв. Рис. 63. Табл. 23.

УДК 621.391:004.94(075.8)
ББК 32.841-в6я73

ISBN 978-5-7996-2503-0

Оглавление

Введение

Разнообразие мира состоит в большом числе составляющих его объектов и многообразии связей между ними. При этом уровень взаимных связей в некоторой системе имеет определяющее влияние на ее поведение, т. е. на число возможных состояний. Поэтому описание и последующий анализ систем даже с небольшим количеством элементов, но «богатым» набором вариантов связей, представляют серьезные математические трудности, иногда непреодолимые. Достаточно вспомнить о задаче коммивояжера [1].

Для описания связей между элементами *дискретных* систем используют разнообразные топологические методы. Одним из таких методов является сетевой подход, при котором наличие и уровень взаимодействия элементов представляется графами и матрицами. Сетевые модели в форме графов получили широкое распространение в различных технических и экономических задачах благодаря дополнительным возможностям, которые появляются при геометрическом подходе к описанию и анализу происходящих в системах процессов. В отличие от евклидовых, прямоугольных, криволинейных и других пространств, в графовых моделях используются концепции топологических геометрий и пространств, отображающих с помощью ветвей связи между вершинами графов, причем форма и длина ветвей, а также местоположение вершин не имеет значения. Перевод топологической информации на понятный ЭВМ язык цифр выполняется с помощью матриц различной размерности.

Таким образом, порядок разработки модели некоторой дискретной системы выглядит следующим образом:

- сначала структура системы визуально представляется графом, имеющим конечное множество вершин и ветвей, снабженных наборами свойств;
- затем представленная графом топологическая информация о системе формализуется в виде матриц, структур и функций, которые запоминаются в компьютере;

- далее в компьютере записывается основная функция (процедура), содержащая определение условий функционирования дискретной системы в виде логико-алгебраических соотношений: равенств (балансов), неравенств (условий), поисков экстремумов и т. п.;
- наконец, подбирается вычислительный метод или программный инструмент, позволяющий за приемлемое время провести численный анализ созданной в компьютере модели дискретной системы.

Современные программные инструменты, как правило, имеют возможности объединять указанные выше этапы разработки и исследования сетевых моделей, что ускоряет получение адекватных результатов анализа и существенно снижает число ошибок пользователя. Однако и сами дискретные системы, и задачи, с ними связанные, настолько индивидуальны, что универсальных программных инструментов для синтеза и анализа сетевых моделей не существует.

Например, рассмотренный в первой главе класс потоковых задач связан с мирной транспортировкой грузов (логистикой) или трубопроводными системами. Но эти же процедуры — связи между максимальными потоками и минимальными сечениями — могут найти и военное применение: построение эффективных планов бомбардировок системы транспортного сообщения противника. Помимо этого с практической точки зрения была бы крайне важна задача об эвакуации на случай бомбардировок или чрезвычайных происшествий. Среди потоковых задач могут быть и простые, например, задачи о назначениях — обобщение задачи о максимальном паросочетании. Модели более сложных задач опираются на теорию динамических потоков. В домашинную эпоху для решения потоковых задач было предложено много «ручных» алгоритмов (метод ветвей и границ, венгерский метод и т. п.), в настоящее время они заменены моделями на основе линейного программирования. Поэтому большинство потоковых задач довольно эффективно решается на компьютерах итерационными методами, в частности с помощью системы моделирования MATLAB.

Во второй главе рассматривается другая сетевая модель, предоставляющая возможности для описания одновременного прохождения информационных и служебных потоков данных в системе по ее различным фрагментам. Второй особенностью этой модели является учет причинно-следственных связей при передаче информации от одного

объекта системы к другому. Для описания логики работы подобной сетевой модели приходится использовать более сложные графические средства описания параллельных процессов — сети Петри. В главе дано краткое описание графического инструментария простейших сетей Петри, а также их более универсальных модификаций — временных, «цветных» и вложенных сетей Петри. Также приводятся сведения о программных инструментах, позволяющих создавать и проводить имитационное моделирование таких сетевых моделей.

Третья глава содержит основные сведения об использовании формализма *E*-сетей при математическом моделировании сложных параллельных систем. Логико-математическое описание содержательного характера строения и взаимодействия объектов этих систем представляется в виде «вход — состояние — выход». Графический аппарат *E*-сетей, во многом схожий с графовыми представлениями сетей Петри, наиболее подходит для описания математических моделей параллельных систем. Функции преобразований объектов легко описываются аппаратом *E*-сетевого моделирования, где параметрическая часть, отражающая хранение и передачу информации, представляется позициями, а математическая, отражающая преобразование параметрической части, представляется переходами.

В четвертой главе учебного пособия описан оригинальный вариант применения сетей компараторов для сортировки данных. В отличие от программно реализуемых процедур, обрабатывающих данные последовательно, указанные сети могут выполнять операции сравнения параллельно. Естественно, они проделывают это за счет усложнения аппаратуры, т. е. связывания групп компараторов сетями по определенному алгоритму. Сеть компараторов может работать параллельно, благодаря этому удается отсортировать n чисел за время $O(\log^2(n))$, что существенно меньшее n тактов.

Пятая глава пособия посвящена нейронным сетям и их применению в обработке сигналов и изображений. Кроме того, нейронные сети могут широко применяться в качестве аппроксимирующих моделей, когда, например, вся нейронная сеть является обобщающей функцией, которая обрабатывает данные, поступающие на ее вход, и выдает соответствующий результат на выходе. В рамках главы проводится краткий обзор различных нейросетевых архитектур и формирование методического материала по использованию нейронных сетей, который может быть полезен для студентов специальностей, связанных

с информационными технологиями. Программным инструментом для моделирования нейронных сетей предлагается использовать приложение nnet системы MATLAB. Использование нейронных сетей в качестве обобщающих функций на сегодняшний день весьма актуально, поскольку при современных вычислительных ресурсах можно строить нейронные сети, которые в полной мере могут реализовать потенциал нейронных сетей в конкретной сфере обработки информации для широкого спектра задач.

Продолжая традиции первых выпусков, в настоящем пособии на базе универсальной системы моделирования MATLAB приведен ряд программных и графических реализаций математических моделей сетевых систем.

Каждая глава сопровождается контрольными вопросами для проверки усвоения изученного учебного материала, а также упражнениями, расширяющими и дополняющими методики моделирования, изложенные в основной части главы.

1. Моделирование транспортных сетей

Сетевые модели в форме графов получили широкое распространение в различных технических и экономических задачах благодаря дополнительным возможностям, которые появляются при геометрическом подходе к описанию и анализе происходящих в системах процессов. О значении геометрической трактовки различных явлений и процессов, в том числе физических, написано большое количество статей и монографий [1, 2], поэтому подробно на этом вопросе останавливаться не будем. Отметим только, что в отличие от евклидовых, прямоугольных, криволинейных и других пространств, в графовых моделях используются концепции топологических геометрий и пространств, отображающих с помощью ветвей связи между вершинами графов, причем форма и длина ветвей, а также местоположение вершин не имеет значения.

Из класса графовых моделей будут рассматриваться только потоковые модели, часто называемые транспортными сетями. Название это обусловлено тем обстоятельством, что данный класс задач возник первоначально при решении транспортных задач, связанных с перевозкой товаров. Под транспортной сетью понимается плоский граф вида, представленного на рис. 1.1, в котором отсутствуют петли, есть начальная вершина (исток) и конечная вершина (сток).

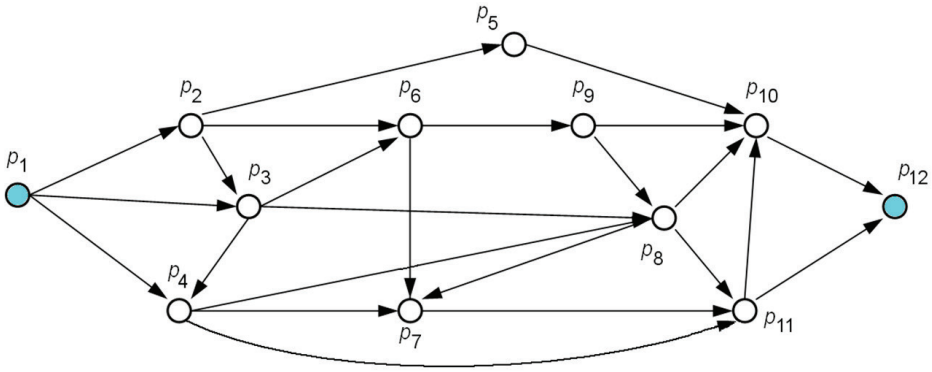


Рис. 1.1. Граф транспортной сети

1.1. Основные понятия и определения теории потоков

Дадим основные понятия теории потоков на графах, которые называются транспортными сетями, в дальнейшем изложении просто сетями.

Конечный ориентированный граф (P, T) *без петель*, имеющий N вершин и M ветвей, называется сетью, если каждой его ветви t_{ij} сопоставлено неотрицательное число $c(t_{ij}) = c_{ij} \geq 0$, называемое *пропускной способностью* ветви. В этом определении ветвь t_{ij} соединяет вершину p_i с вершиной p_j. Такой граф обладает следующими свойствами:

1) существует одна и только одна вершина p_1, из которой ветви выходят, но ни одна ветвь в нее не входит. Эта вершина называется входом или *истоком* сети;
2) существует одна и только одна вершина графа p_N, в которую входят ветви, но ни одна ветвь из нее не выходит. Эта вершина называется выходом или *стоком* сети.

Если две какие-либо вершины (p_i и p_j) соединены непосредственно несколькими параллельными ветвями, то на место этих ветвей вставляется одна обобщенная ветвь t_{ij} с суммарной пропускной способностью c_{ij}. Ориентация обобщенной ветви зависит от знака алгебраической суммы c_{ij}. В некоторых случаях при преобразовании сети наряду с подобным объединением может быть применен процесс расщепления одиночных ветвей на несколько параллельных ветвей при условии, что суммарная пропускная способность остается неизменной.

Для сетей вводят понятие потока. Пусть T_i^+ — множество ветвей, входящих в вершину p_i, а T_i^- — множество ветвей, выходящих из вершины p_i. Функция $\varphi(t)$, определенная на множестве ветвей сети и принимающая положительные значения, представляет собой поток данной транспортной сети, если

$$\varphi(t_{ij}) \leq c(t_{ij}), \tag{1.1}$$

$$\sum_{t \in T_i^-} \varphi(t) = \sum_{t \in T_i^+} \varphi(t); \; i \neq 1, \; i \neq N. \tag{1.2}$$

Условие (1.1) означает, что поток ветви не может превышать ее пропускную способность. По условию (1.2) суммарный поток входящих в вершину ветвей равен суммарному потоку выходящих ветвей (за ис-

ключением вершин входа и выхода). Иногда последнее соотношение называется условием сохранения потока, из него следует, что в любой промежуточной вершине сети поток не создается и не исчезает. Иными словами, выполняется закон сохранения в отношении потоков в вершинах как аналог закона Кирхгофа для токов в узле электронной схемы.

Поток в сети, с формальной точки зрения, это неотрицательная вещественнозначная функция, определенная на ветвях сети, обладающая дополнительными условиями консервативности (для каждой нетерминальной вершины сумма потока по входящим ребрам равна сумме потока по исходящим) и подчиненности пропускным способностям (поток по ребру не превышает его пропускной способности).

Очевидно, что транспортные сети типа сети автомобильных дорог или железнодорожных путей, сети линий связи (телефонных, телевизионных или компьютерных) в большинстве случаев удовлетворяют условиям, накладываемым на транспортную сеть в теории потоков. Ветви можно интерпретировать как пути сообщения; их пропускная способность $c(t)$ на практике может, например, означать *максимальное* количество товаров или материалов, которые могут быть перевезены по участку дороги в единицу времени. В этом случае значение потока на ветви $\varphi(t)$ представляет собой количество материалов, которое *действительно* перевозится по дорожному участку в единицу времени.

Очевидно, что любая задача, определенная на конечном графе, может быть решена хотя бы просто с помощью простого перебора. Однако метод полного перебора весьма затратен и на практике почти не используется.

Введем понятие суммарного потока сети Φ на конечных ветвях (конечные ветви — это ветви, примыкающие либо только к истоку, либо только к стоку), отличное от понятия потока на ветви, которое рассматривалось ранее. Из уравнения баланса (1.2) следует, что сумма потоков, исходящих из начальной вершины (истока) p_1 (рис. 1.2), равна сумме потоков, входящих в конечную вершину (сток) p_N, т. е.

$$\sum_{t \in T_1^-} \varphi(t) = \sum_{t \in T_N^+} \varphi(t) = \Phi. \qquad (1.3)$$

Соотношения (1.1) и (1.3) определяют функцию $\varphi(t)$, называемую потоком в сети (P, T) с суммарным потоком на конечных ветвях Φ, который ставит в соответствие каждой ветви сети определенное положительное число $\varphi_{ij} = \varphi(t_{ij})$, $i, j = 1, 2, \ldots, N$. Это последнее число ино-

гда называют потоком ветви, т. е. когда говорят, что задан поток $\varphi(t)$ в сети, то это значит, что заданы значения потока на всех ее M ветвях. Поток сети можно представить как M-мерный вектор с неотрицательными элементами.

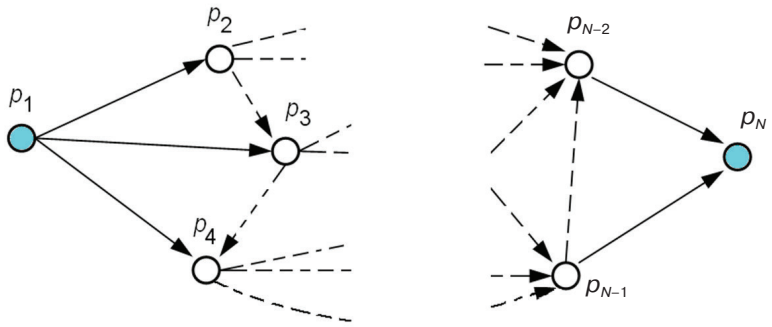


Рис. 1.2. Конечные ветви сети

Таким образом, каждая ветвь транспортной сети имеет два атрибута: первый — потенциальный — это ее пропускная способность, второй — актуальный — это поток, проходящий через ветвь:

$$t_{ij} = (c_{ij}, \varphi_{ij}),\ T = \{t_{ij}\} = \{(c_{ij}, \varphi_{ij})\},\ i, j = 1,..., N, \qquad (1.4)$$

где $\varphi_{ij} = \varphi(t_{ij})$ — поток ветви t_{ij}. Например, на рис. 1.3, *а* представлена сеть с четырьмя вершинами. Около каждой ветви проставлена пропускная способность (первое число) и значение потока на ветви (второе число, взятое в круглые скобки). Значение суммарного потока на конечных ветвях $\Phi = 6$.

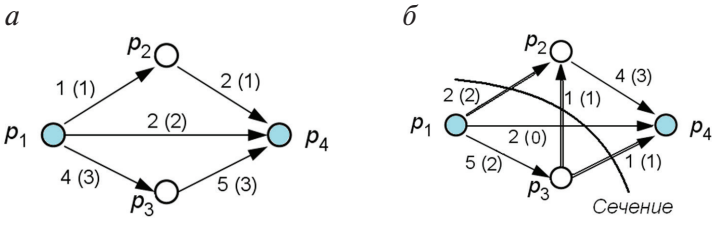


Рис. 1.3. Транспортная сеть:
а — определение потока; *б* — определение сечения

Перейдем к определению понятия канонического сечения сети. Вообще *сечением* называется множество ветвей графа, при удалении которых он разбивается на два несвязных подграфа. Допустим, что

множество всех вершин сети разбито на два взаимно дополнительных подмножества P_1 и P_N, причем первое подмножество содержит вход сети $p_1 \in P_1$, а второе — выход сети $p_N \in P_N$, т. е. все вершины сети, которые не вошли в подмножество P_1, должны содержаться в подмножестве P_N: $P_1 \cap P_N = 0$; сумма двух подмножеств P_1 и P_N составляет множество всех вершин сети: $P = P_1 \cup P_N$. В этом случае говорят, что в сети выполнено сечение, и множество $T_{1N} = \{t_{ij}\} = \{t(p_i \to p_j)\}$ ветвей сети, где $p_i \in P_1$, $p_j \in P_N$, называется каноническим сечением.

Сумма пропускных способностей ветвей сечения называется пропускной способностью сечения и обозначается Г и, например, для канонического сечения

$$\Gamma_{1N} = \Gamma(T_{1N}) = \sum_{T_{1N}} c_{ij}. \tag{1.5}$$

В последнем обозначении указываются взаимно дополнительные подмножества, определяющие каноническое сечение. Как правило, с изменением сечения изменяется и его пропускная способность. Если взять карту автомобильных дорог или железнодорожных путей и с помощью ножниц разрезать ее на две части так, чтобы разрез не попал на города (вершины), то получатся два взаимно дополнительных множества городов, определяющих некоторое сечение. Аналогичный «эксперимент» можно проделать, если разрезать все телефонные провода, связывающие населенные пункты Московской области с населенными пунктами Ленинградской области.

В общем случае величина потока на конечных ветвях никогда не превышает пропускную способность канонического сечения:

$$\Phi \le \Gamma_{1N} = \sum_{T_{1N}} c_{ij}. \tag{1.6}$$

Отметим одно очень важное для дальнейшего изложения обстоятельство. Если в соотношении (1.6) имеет место равенство, то Ф является наибольшим потоком, а Г — наименьшей пропускной способностью сечения, допустимыми в графе. Это следует из того факта, что в соответствии с соотношением (1.6) величина пропускной способности разреза Г не может принимать значение, меньшее максимального среди возможных значений потока Ф; а с другой стороны, величина Ф не может превышать наименьшее значение Г.

Следует подчеркнуть, что если найдутся такие два значения величин Ф и Г, которые равны между собой, то эти значения являются соответственно максимально и минимально возможными значениями. Одна-

ко отсюда пока еще не следует, что в любом графе $\Phi_{max} = \Gamma_{min}$, так как в определении имеется оговорка: «если найдутся такие два значения».

Фордом и Фалкерсоном доказана фундаментальная теорема [3], имеющая чрезвычайно большое значение в теории графов: *в любой сети максимальное значение суммарного потока на конечных ветвях равно минимальной пропускной способности канонического сечения*.

На рис. 1.3, *б* представлена сеть, состоящая из четырех вершин и шести ветвей. Нетрудно убедиться, что условие сохранения потока для промежуточных вершин p_2 и p_3 удовлетворяется. Значение потока сечения $\Phi = 4$. Можно найти сечение, пропускная способность которого $\Gamma = 4$. На рисунке это сечение, состоящее из ветвей t_{12}, t_{32}, t_{34}. Из этого примера следует, что сечение не всегда совпадает с понятием «разреза с помощью ножниц», так как в этом частном случае в сечение не включена ветвь t_{14}, поскольку поток $\varphi_{14} = 0$. На основании ранее сказанного суммарный поток на конечных ветвях в этой сети будет максимальным, а пропускная способность выбранного сечения минимальной. Пример показывает, что максимальный поток сети не обязательно состоит из максимальных значений потока по каждой ветви, т. е. в большинстве случаев ветви транспортной сети «работают» не в полную силу и поток в них часто меньше пропускной способности.

Следствием теоремы Форда и Фалкерсона является утверждение о том, что поток φ является максимальным в том и только в том случае, если нет ни одного пути, увеличивающего его.

Это следствие дает эффективный способ определения максимального потока, который будет использован в дальнейшем. Предположим, что в сети выбрано каноническое сечение $T_{1N} = T_1 \cup T_N$, где T_1 и T_N – множества ветвей, относящиеся к истоку и стоку соответственно. При этом все ветви, входящие в множество T_1, выбираются ориентированными в сторону стока (положительные ветви в T_1), а ветви, входящие во множество T_N, выбираются ориентированными в противоположном направлении (отрицательные ветви в T_N). Для того чтобы поток через сечение T_{1N} сделать максимальным, необходимо «насытить» все его ветви. Ветвь t_{ij} считается насыщенной, если $\varphi(t_{ij}) = c(t_{ij})$, и свободной от потока, если $\varphi(t_{ij}) = 0$. Ветвь, одновременно насыщенная и свободная от потока, имеет нулевую пропускную способность $c(t_{ij}) = 0$. Отсюда следует, что поток через сечение T_{1N}^1 максимален в том и только в том случае, если каждый максимальный поток φ насыщает все ветви множества T_1 и оставляет свободными все ветви, принадлежащие множеству T_N.

Равенство (1.3) можно считать основным условием *сбалансированной транспортной сети*: сумма потоков, исходящих из истока, равна сумме потоков, входящих в сток. Дополнительным условием является равенство сумм потенциальных способностей конечных ветвей: сумма потенциальных способностей истоковых ветвей равна сумме потенциальных способностей стоковых ветвей. Последнее условие используется при построении сбалансированной транспортной сети и превращается в основное при превышении или равенстве пропускной способности канонического сечения суммы потенциальных способностей конечных ветвей (см. неравенство (1.6)).

Неравенство сумм потенциальных способностей конечных ветвей $\sum_{\{j\in T_1\}} c_{1j} = \Gamma_1 \neq \sum_{\{i\in T_N\}} c_{iN} = \Gamma_N$ можно назвать потенциальной разбалансированностью транспортной сети, неравенство сумм реальных потоков конечных ветвей $\sum_{\{j\in T_1\}} \varphi(t_{1j}) = \Phi_1 \neq \sum_{\{i\in T_N\}} \varphi(t_{iN}) = \Phi_N$ — ее реальной разбалансированностью. Анализ разбалансированных сетей легко сводится к анализу сбалансированных сетей с помощью простых добавлений двух вершин: одной новой конечной, другой дополнительной внутренней. При этом старая конечная вершина также становится внутренней, а две дополнительные ветви в сумме балансируют конечные потенциальные способности (рис. 1.4).

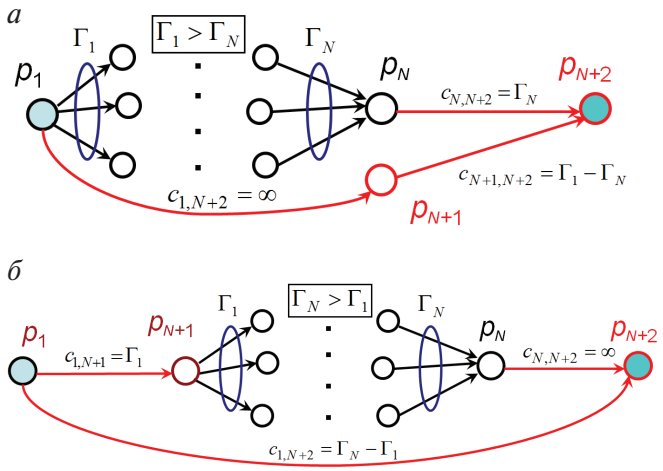


Рис. 1.4. Балансировка транспортной сети:

а — при интенсивности истока больше интенсивности стока; *б* — при интенсивности стока больше интенсивности истока

1.2. Максимальный поток в транспортной сети

Задача о максимальном потоке изучается сравнительно недавно. Интерес к ней обусловлен широким практическим применением — на коммуникационных, транспортных, электрических сетях, при моделировании различных процессов физики и химии, в некоторых операциях над матрицами, для решения родственных задач теории графов и даже для поиска Web-групп в Интернете. Первое решение основывалось на simplex-методах линейного программирования, что было крайне неэффективно. Форд и Фалкерсон предложили рассматривать для решения задачи о максимальном потоке ориентированную сеть и искать решение с помощью итерационного алгоритма. В течение 20 лет все передовые достижения в исследовании данной задачи базировались на их методе. В 1970 г. наш соотечественник Е. А. Диниц предложил решать задачу с использованием вспомогательных бесконтурных сетей и псевдомаксимальных потоков, что намного увеличило быстродействие разрабатываемых алгоритмов. В 1974 г. А. В. Карзанов улучшил метод Диница, введя такое понятие, как предпоток. Алгоритмы Диница и Карзанова, как и исследования Форда и Фалкерсона, внесли огромный вклад в решение данной проблемы. На основе их методов в течение 15 лет достигались наилучшие оценки быстродействия алгоритмов. В 1986 г. появился третий метод, который также можно отнести к фундаментальным. Этот метод был разработан А. В. Голдбергом и Р. Е. Таряном и получил название Push-Relabel (PRM). Для нахождения максимального потока он использует предпотоки и метки, изменяемые во время работы алгоритма. PRM-алгоритмы очень эффективны и исследуются до сих пор. И, наконец, в 1997 г. А. В. Голдберг и С. Рао предложили алгоритм, присваивающий дугам неединичную длину. Это самый современный из известных на сегодня алгоритмов. Асимптотическая оценка его быстродействия превзошла $O(nm)$, где n и m — количество вершин и ветвей графа сети соответственно.

Данциг и Фалкерсон показали, что теорема о максимальном потоке и минимальном сечении может быть выведена из теоремы о сильной двойственности задач линейного программирования. Более того, было доказано существование максимального целочисленного потока для сетей, пропускные способности которых являются целочисленными.

Рассмотрим математическую модель максимального потока в терминах линейного программирования, т. е. уравнения, учитывающие балансы в вершинах и условия, накладываемые на потоки ветвей. Для этого нам потребуются знания о свойствах прямой и двойственной задач линейного программирования.

1.2.1. Алгоритм линейного программирования

Традиционно прямая задача формулируется следующим образом [4]. Требуется определить значения переменных $x_1, x_2, ..., x_n$, которые удовлетворяют соотношениям

$$\begin{cases} a_{11}x_1 + \ldots + a_{1k}x_k + \ldots + a_{1n}x_n = b_1, \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ a_{k1}x_1 + \ldots + a_{kk}x_k + \ldots + a_{kn}x_n = b_k, \\ a_{k+1,1}x_1 + \ldots + a_{k+1,k}x_k + \ldots + a_{k+1,n}x_n \le b_{k+1}, \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ a_{m1}x_1 + \ldots + a_{mk}x_k + \ldots + a_{mn}x_n \le b_m \end{cases} \tag{1.7}$$

и обращают в минимум линейную форму

$$f_1x_1 + \ldots + f_kx_k + \ldots + f_nx_n = f^{\mathrm{T}}x = L. \tag{1.8}$$

Для дальнейшего важно, что величины $x_1, x_2, ..., x_k$, относящиеся к равенствам (их ровно k), имеют любые знаки, а величины, относящиеся к неравенствам, неотрицательны или равны нулю: $x_{k+1}, x_{k+2}, \ldots, x_n \ge 0$ (их ровно $l = n - k$). Искомые переменные $x_1, x_2, ..., x_n$ можно представить как потоки всех n ветвей, первые k уравнений системы (1.7) как уравнения баланса потоков для k вершин, а неравенства системы как ограничения на потоки ветвей со стороны их пропускных способностей. При этом минимум (или максимум с обратным знаком) линейной формы должен относиться к суммарному потоку на обоих граничных вершинах.

На рис. 1.5 показана структура системы уравнений и неравенств (1.7). При $m = k$ она вырождается в обычную систему линейных уравнений, имеющую (при определителе системы, не равном нулю) единственное решение x_{eq}. В этом случае поиск минимума линейной формы (1.8) при заданном наборе коэффициентов f становится бессмысленным:

область ее поиска — единственное число. Наличие $(m - k) > 0$ неравенств существенно расширяет область поиска минимума и тем самым определяет задачу линейного программирования.

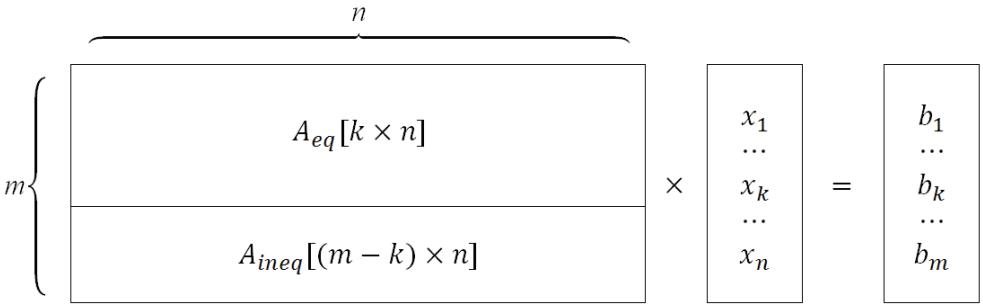


Рис. 1.5. Графический образ системы (1.7)

Этой прямой задаче соответствует двойственная задача. При этом каждому из уравнений и неравенств (1.8) ставятся в соответствие двойственные переменные $y_1, y_2, ..., y_m$ и для этих переменных составляются ограничения, состоящие из равенств и неравенств

$$\begin{cases} a_{11}y_1 + \ldots + a_{l1}y_l + \ldots + a_{m1}y_m = f_1, \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ a_{1l}y_1 + \ldots + a_{ll}y_l + \ldots + a_{ml}y_m = f_l, \\ a_{1,l+1}y_1 + \ldots + a_{l,l+1}y_l + \ldots + a_{m,l+1}y_m \geq f_{l+1}, \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ a_{1n}y_1 + \ldots + a_{ln}y_l + \ldots + a_{lm}y_m \geq f_n, \end{cases} \tag{1.9}$$

причем переменные $y_1, y_2, ..., y_l$ имеют произвольные знаки, а остальные — неотрицательные значения $y_{l+1}, y_{l+2}, \ldots, y_m \geq 0$. Требуется определить такие значения $y_1, y_2, ..., y_m$, которые удовлетворяют соотношениям (1.9) и максимизируют линейную форму

$$b_1y_1 + \ldots + b_ly_l + \ldots + b_my_m = b^T y = U. \tag{1.10}$$

Коэффициенты в левой части (матрица ограничений A^{T}) двойственной задачи (1.9) получаются из транспонированной матрицы коэффициентов левой части (матрица ограничений A) прямой задачи (1.7). Количество равенств в системе уравнений (1.9) равно числу неравенств в системе (1.7), а сами эти равенства соответствуют $x_1, x_2, ..., x_l$, имею-

щим любые знаки, а неравенства — неотрицательным переменным $x_{l+1}, x_{l+2}, \ldots, x_n$.

Вектор $x = [x_1; x_2; \ldots, x_n]^{\mathrm{T}}$, удовлетворяющий ограничениям основной задачи, называется *допустимым* вектором, а основная задача — *допустимой задачей*. Допустимый вектор, максимизирующий линейную форму (1.8), называется *оптимальным*. Такая же терминология применяется для двойственной задачи, т. е. для вектора $y = [y_1; y_2; \ldots, x_m]^{\mathrm{T}}$. Для каждой из этих задач возможен один из трех случаев:

1) имеются оптимальные векторы (а следовательно, и допустимые);

2) имеются допустимые векторы, но не имеется оптимальных;

3) не имеется допустимых векторов.

Основная теорема двойственности линейного программирования утверждает [1, 4], что если одна из этих ситуаций имеет место для основной задачи, то для двойственной задачи существует та же самая ситуация, и наоборот. Кроме того, минимум L в основной задаче равен минимуму U в двойственной, т. е.

$$\min L = \max U, \tag{1.11}$$

если для основной задачи, а следовательно, и для двойственной имеет место случай 1.

По второй теореме двойственности для оптимальности допустимых решений x_d и y_d необходимо и достаточно выполнение следующего условия: для того чтобы какое-либо неравенство системы ограничений одной задачи не обратилось в точное равенство, соответствующая переменная другой задачи должна равняться нулю.

Иная формулировка этого же утверждения: для того чтобы допустимые решения x_d и y_d пары задач были оптимальными, необходимо и достаточно выполнение следующего условия: если какая-либо переменная одной двойственной задачи строго положительна, то соответствующее ограничение другой задачи должно обратиться в равенство.

Пример 1.1. Линейное программирование в системе MATLAB.

Линейное программирование в системе MATLAB определено как [5]

$$\min f^{\mathrm{T}} x \text{ при решении } \begin{pmatrix} A_{eq} \cdot x = b_{eq} \\ A_{ineq} \cdot x \le b_{ineq} \\ l \le x \le u \end{pmatrix}. \tag{1.12}$$

Реализация этого метода основана на модуле LIPSOL [6], который является вариантом алгоритма корректора-предсказателя Мехротры [7] (Mehrotra's predictor-corrector) прямого и двойственного метода внутренней точки (primal-dual interior-point method).

Алгоритм начинается с ряда шагов предварительной обработки. Ее результатом является выполнение следующих требований:

- все переменные ограничены снизу нулем;
- все ограничения преобразованы в равенства;
- переменные-константы, т. е. переменные, равные верхним или нижним граничным значениям, удалены;
- нулевые столбцы в матрице ограничения удалены;
- нулевые строки в матрице ограничения удалены;
- матрица ограничения имеет полный ранг (по строкам и по столбцам);
- если в матрице ограничения *имеется* существенное количество строк с одинаковыми связанными переменными, то они разрешаются и удаляются.

Эти шаги предварительной обработки могут много сделать для ускорения выполнения повторяющейся части алгоритма, но при этом требуется обращаться к множителям Лагранжа. Поэтому результаты представленной выше предварительной обработки могут быть проигнорированы, так как множители в ходе выполнения алгоритма в любом случае рассчитываются для преобразованной задачи, а не оригинала. Однако, если обращение к множителям Лагранжа пользователь проводит, то это преобразование не отбрасывается и может сэкономить некоторое время вычислительного процесса.

После предварительной обработки задача имеет форму, в которой неравенства преобразованы в равенства:

$$\min f^{\mathrm{T}}x \text{ при решении} \begin{pmatrix} A \cdot x = b, \\ 0 \le x \le u \end{pmatrix}. \tag{1.13}$$

При этом верхние границы неявно включены в матрицу системы A. С введением добавочных (свободных от англ. *slack* = зазор, люфт) переменных s система (1.13) примет вид:

$$\min f^{\mathrm{T}}x \text{ при решении} \begin{pmatrix} A \cdot x = b, \\ x + s = u, \\ x \ge 0, s \ge 0 \end{pmatrix}. \tag{1.14}$$

Такая форма называется *прямой задачей*: вектор x состоит из основных переменных, а вектор s — из добавочных переменных.

Двойственная задача линейного программирования

$$\max (b^{\mathrm{T}}y - u^{\mathrm{T}}w) \text{ при решении } \underset{z\geq 0,\, w\geq 0}{A^{\mathrm{T}} \cdot y - w + z = f}. \qquad (1.15)$$

где y и w состоят из двойственных переменных, а z состоит из двойственных добавочных переменных. Оптимальные соотношения для этой линейной задачи, т. е. для прямой (1.14) и для двойственной (1.15) одновременно, примут вид:

$$F(x,y,z,s,w) = \begin{pmatrix} A\cdot x - b, \\ x + s - u, \\ A^{\mathrm{T}}\cdot y - w + z - f, \\ x_i z_i, \\ s_i w_i, \end{pmatrix} = 0 \text{ при } x \geq 0, z \geq 0, s \geq 0, w \geq 0, \quad (1.16)$$

где $x_i z_i$ и $s_i w_i$ означают поэлементное умножение.

В линейном программировании квадратичные уравнения $x_i z_i = 0$ и $s_i w_i = 0$ называются условиями *взаимозависимости*; другие (остальные) линейные уравнения называются условиями *выполнимости*. Величина $x^{\mathrm{T}}z + s^{\mathrm{T}}w$ является *дуальной разностью*, которая служит мерой ненулевого остатка в зависимости F, когда $(x,z,s,w) \geq 0$.

Прямой-двойственный алгоритм (*primal-dual algorithm*) означает, что и основная и двойственная задачи решаются одновременно. Он похож на метод Ньютона: при решении в ходе выполнения циклов переменные x, z, w и s остаются положительными, откуда произошло дополнение в его название — метод внутренней точки. Повторение проводится в строго внутренней области, представленной ограничениями в виде неравенств в задаче (1.14) линейно-квадратичной системы $F(x,y,z,s,w) = 0$ в задаче (1.16).

Численный алгоритм — вариант алгоритма корректора-предсказателя — предложен (как сказано выше) Мехротрой. Рассмотрим процедуру циклических вычислений $v = [x; y; z; s; w]$ при стремлении ошибки к нулю: $0 \to [x; y; z; s; w] > 0$. Сначала вычисляется так называемая величина *предсказания*

$$\Delta v_p = -(F^{\mathrm{T}}(v))^{-1} F(v), \qquad (1.17)$$

которая является невязкой в методе Ньютона; потом — так называемая величина корректора

$$\Delta v_c = -(F^{\mathrm{T}}(v))^{-1}(F(v+\Delta v_p)) - \mu\delta, \qquad (1.18)$$

где $0 \to \mu > 0$ назван параметром *сосредоточения* и должен быть тщательно выбран. Параметр δ является вектором, состоящим из нулей и единиц, причем единицы находятся на тех местах, на которых в $F(v)$ расположены квадратичные уравнения. Таким образом коррекция применяется только к условиям взаимозависимости, которые все являются квадратичными, но не к условиям выполнимости, которые все линейны. Эти две величины объединяются на каждом итерационном шаге с помощью параметра $0 \to \alpha > 0$ для получения по известному с предыдущего шага v нового значения v^+:

$$v^+ = v + \alpha(\Delta v_p + \Delta v_c), \qquad (1.19)$$

где параметр α на каждом шаге выбран таким образом, чтобы $v^+ = [x^+; y^+; z^+; s^+; w^+]$ удовлетворяло требованию $0 \to [x^+; y^+; z^+; s^+; w^+]$, т. е. чтобы $\left|F(v^+)\right| < \left|F(v)\right|$.

В ходе вычислений на предыдущих шагах алгоритм вычисляет (для матрицы A в разреженном виде) прямое разложение Холесского матрицы системы A на множители в виде $A = A_l A_l^{\mathrm{T}}$. Если матрица A имеет хорошо заполненные столбцы, то вместо этого разложения используется формула Шермана–Моррисона (*Sherman-Morrison formula*), и если это решение не адекватно (невязки слишком большие), то, чтобы найти численное решение, используют предобусловленные сопряженные градиенты (*preconditioned conjugate gradients*).

Алгоритм повторяет описанные выше шаги до тех пор, пока процесс не сойдется. Основной критерий остановки — стандартный

$$\frac{\|r_b\|}{\max(1,\|b\|)} + \frac{\|r_f\|}{\max(1,\|f\|)} + \frac{\|r_u\|}{\max(1,\|u\|)} + \frac{\left|f^{\mathrm{T}}x - b^{\mathrm{T}}y + u^{\mathrm{T}}w\right|}{\max(1,\left|f^{\mathrm{T}}x\right|,\left|b^{\mathrm{T}}y - u^{\mathrm{T}}w\right|)} \le tol, \qquad (1.20)$$

где $r_b = Ax - b$, $r_f = A^{\mathrm{T}}y - w + z - f$, $r_u = x + s - u$ есть прямая невязка, двойственная невязка и величина отклонения от верхней границы соответственно, а значение соотношения $f^{\mathrm{T}}x - b^{\mathrm{T}}y + u^{\mathrm{T}}w$ является разностью между реальными значениями прямой и двойственной задач. Задаваемый пользователем параметр *tol* есть некоторая положительная по-

грешность решения оптимизационной задачи. Итоговая сумма критерия остановки означает полную относительную ошибку при заданных условиях оптимальности в системе (1.16).

1.2.2. Математические модели максимального потока

Теорему о максимальном потоке можно получить из теоремы двойственности. Теорема двойственности, теорема о максимальном потоке и теорема о минимаксе в теории игр — это различные формулировки одного и того же положения [1]. В дальнейшем часто будет использоваться переход к двойственной задаче. Рассмотрим уравнения, определяющие максимальный поток, и составим для них прямую и двойственные задачи. Прямая задача формулируется достаточно просто, поскольку непосредственно связана с потоками ветвей и очевидными ограничениями на их величину:

$$\sum_{j\in T_i^-}\varphi(t_{ij})-\sum_{j\in T_i^+}\varphi(t_{ji})=\begin{cases}-\Phi, & i=1;\\ 0, & i\neq 1,N;\\ \Phi, & i=N;\end{cases} \tag{1.21}$$

$$\varphi(t_{ij})\le c(t_{ij}),\quad i,j\in\overline{1,N}; \tag{1.22}$$

$$\Phi\to\max. \tag{1.23}$$

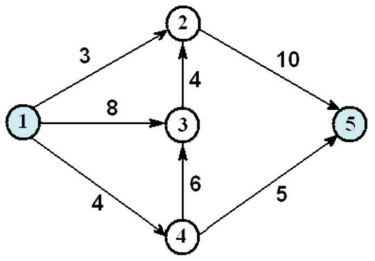
Хотя поиск экстремума в выражении (1.23) направлен в сторону максимума (как в двойственной задаче), смена знака в этом соотношении приводит к эквивалентному поиску минимума (как в прямой задаче). Нетрудно убедиться в том, что матрица коэффициентов левой части уравнения (1.21) совпадает с матрицей инциденций графа транспортной сети. Поясним это на примере.

Пример 1.2. Прямая задача линейного программирования для транспортной сети

Для сети, изображенной на рис. 1.6, *а*, матрица инциденций имеет вид, представленный на рис. 1.6, *б*. Строки матрицы инциденций соответствуют вершинам графа, столбцы — его ветвям. Если ветвь выходит из вершины, то на пересечении соответствующих строки и столбца стоит −1, если входит, то стоит 1. Когда вершина и ветвь неинцидент-

ны, т. е. вершина не является ни начальной, ни конечной точкой ветви, соответствующий элемент матрицы равен нулю.

а



б

	t_{12}	t_{13}	t_{14}	t_{43}	t_{32}	t_{25}	t_{45}
p_1	–1	–1	–1	0	0	0	0
p_2	1	0	0	0	1	–1	0
p_3	0	1	0	1	–1	0	0
p_4	0	0	1	–1	0	0	–1
p_5	0	0	0	0	0	1	1

Рис. 1.6. Граф транспортной сети — *а* и его матрица инциденций — *б*

Обозначим вектор искомых переменных x и сопоставим его первые семь элементов с неизвестными потоками, проходящими через каждую из семи ветвей транспортной сети:

$$x_1 = \varphi(t_{12}),\quad x_2 = \varphi(t_{13}),\quad x_3 = \varphi(t_{14}),\quad x_4 = \varphi(t_{43}),$$
$$x_5 = \varphi(t_{32}),\quad x_6 = \varphi(t_{25}),\quad x_7 = \varphi(t_{45}).$$

Далее, включим в состав вектора x неизвестную величину потока стока Ф (и истока), добавив его в качестве восьмого элемента: $x_8 = \Phi$.

Согласно системе (1.21) балансов вершин составим систему линейных равенств, матрицей которой является матрица инциденций с дополнительным столбцом, относящимся к восьмому элементу вектора x:

$$A_{eq}x = \begin{bmatrix} -1 & -1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Первая строка матрицы A_{eq} относится к истоку и означает, что весь поток, выходящий из него, точно распределен по трем ветвям, выходящим из этой вершины (в последний столбец добавлена 1). То же справедливо в отношении последней строки матрицы, определяющей

взаимосвязь всех входящих в сток ветвей и потока стока (в последний столбец добавлена –1). Остальные строки соответствуют строкам матрицы инциденций с добавлением нуля в последний столбец (алгебраическая сумма потоков каждой вершины равна нулю).

Система линейных неравенств (1.22), соответствующая средней строке уравнения (1.12), определяет ограничения на величину потока каждой ветви, накладываемые их пропускными способностями:

$$A_{ineq}x = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} \leq \begin{bmatrix} c_{12} \\ c_{13} \\ c_{14} \\ c_{43} \\ c_{32} \\ c_{25} \\ c_{45} \\ 0 \end{bmatrix}$$

Поскольку вектор x содержит восемь элементов, то восьмая строка матрицы неравенств, относящаяся к переменной максимального потока Ф, заполнена нулями, кроме последнего значения, равного –1. При этом для положительной величины потока ограничение сверху снимается и превращается в ограничение снизу величиной, равной 0. Заметим, что элемент $A_{ineq}(8,8)$ можно превратить в 1, если поменять знаки в последнем столбце матрицы A_{eq}. В этом случае максимум потока в сети должен быть найден как минимум его отрицательной величины: $\max \Phi = \min(-\Phi)$.

Итак, прямая задача линейного программирования, составленная для определения максимального потока транспортной сети на рис. 1.6, *а*, имеет вид:

$$\begin{cases} A_{eq}x = 0; \\ A_{ineq}x \leq c; \\ x \geq 0; \\ \max(x_8) = \min(f^{\mathrm{T}}x), \end{cases}$$

где вектор линейной формы прямой задачи есть $f^{\mathrm{T}} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ -1]$, а вектор ее ограничений — $c^{\mathrm{T}} = [3\ 8\ 4\ 6\ 4\ 10\ 5\ 0]$. Решение прямой за-

дачи можно получить, обратившись к математическим программам, однако в этом простом примере оно находится довольно просто «вручную»: максимальный поток равен 11 единицам, а потоки каждой ветви представлены в табл. 1.1.

Таблица 1.1

Потоки ветвей при максимальном потоке сети

Ветвь	t_{12}	t_{13}	t_{14}	t_{43}	t_{32}	t_{25}	t_{45}
Способность	3	8	4	6	4	10	5
Поток	3	4	4	0	4	7	4
КПД ветви	1.00	0.50	1.00	0.00	1.00	0.70	0.80

1.2.3. Математические модели минимального сечения

Рассмотрим формирование двойственной задачи линейного программирования по данным ее прямой задачи. В отличие от прямой задачи, где ищется максимальный поток в сети, здесь ищется минимальное сечение. Сопоставим равенствам (1.21) величины π_i, означающие признак принадлежности (0 или 1) подмножеству P_1 или P_N. Всего этих величин N, где N — число вершин сети. Также сопоставим неравенствам (1.22) величины γ_{ij}, означающие признак принадлежности (0 или 1) множеству ветвей минимального сечения T_{1N} (всего их может быть N^2). Напомним правило перехода: если какая-либо переменная одной двойственной задачи строго положительна, то соответствующее ограничение другой задачи должно обратиться в равенство. Тогда в соответствии с этими правилами получим:

$$\pi_1 + \pi_N \geq 1; \tag{1.24}$$

$$\pi_i - \pi_j + \gamma_{ij} \geq 0, \quad i, j \in \overline{1, N}; \tag{1.25}$$

$$\gamma_{ij} \geq 0, \quad i, j \in \overline{1, N}; \tag{1.26}$$

$$c^T \gamma \to \min. \tag{1.27}$$

Как следует из соотношений (1.7) и (1.9), если в основной задаче имеется k первых соотношений типа равенств, а остальные $n - k$ соотношений имеют тип неравенств со знаком $\leqslant$, то в двойственной задаче получается l первых соотношений типа равенств (l — число

переменных основной задачи, которые могут иметь любые знаки) и $n - l$ остальных (последних) соотношений типа неравенств со знаком $\geqslant$. Дело в том, что в основной задаче, которая задается соотношениями (1.21)—(1.23), все величины неотрицательные (по определению потока $\varphi(t) \geq 0$). В основной задаче нет неизвестных величин, которые имеют произвольный знак. Поэтому в двойственной задаче будут только соотношения вида неравенств ($\geqslant$), а равенства отсутствуют, т. е. $l = 0$.

Группа соотношений (1.24)–(1.25) получается из строк матрицы, транспонированной к матрице (1.21)–(1.23) прямой задачи. Соотношения (1.26) следуют из общего правила составления двойственной задачи, согласно которому двойственные переменные, соответствующие ограничениям основной задачи типа $\leqslant$, неотрицательны. Переменные π_i имеют произвольные знаки, так как они соответствуют равенствам основной задачи.

Линейная форма (1.27) состоит только из множителей c_{ij} и величин γ_{ij}, поскольку соотношение (1.24) для переменной Ф было отнесено к равенствам при транспонировании матрицы прямой задачи. Единица в правой части (1.24) стоит потому, что соответствующий коэффициент линейной формы в системе (1.21) равен единице. Переменная Ф в общем случае имеет последний $[(N+1)N+1]$-й номер, и уравнение двойственной задачи, соответствующее последнему столбцу матрицы $A_{ineg}(8, 8)$, тоже последнее по номеру, хотя в соотношениях (1.21) и (1.24) оно стоит первым.

Оптимальным решением двойственной задачи является T_{1N} — минимальное сечение, отделяющее x_1 от x_N (исток от стока). При этом значения переменных π_i и γ_{ij} могут быть только либо нулями, либо единицами. Если переменная $\pi_i = 1$, то i-я вершина относится к истоку, в противном случае она включается в набор вершин стока. Если переменная $\gamma_{ij} = 1$, то ветвь, соединяющая i-ю вершину с j-й вершиной, включена в минимальное сечение. Все ветви, для которых переменные $\gamma_{ij} = 0$, не входят в минимальное сечение и, следовательно, не оказывают влияния на формирование линейной формы (1.27). Таким образом, она состоит только из суммы пропускных способностей ветвей минимального сечения (см. формулу (1.5)) и, согласно линейной форме (1.10), она равна максимальному потоку транспортной сети.

Решение двойственной задачи дает целочисленный набор $\{\pi_i\}$ — вершин подмножества, определяющего минимальное сечение, и це-

лочисленный набор $\{\gamma_{ij}\}$ — ветвей, соединяющих концевые подмножества минимального сечения.

Таким образом, решение двойственной задачи транспортной сети определяет минимальное сечение, а решение ее прямой задачи — максимальный поток.

Пример 1.3. Двойственная задача линейного программирования для транспортной сети.

Для сети, изображенной на рис. 1.6, *а*, двойственная задача примет вид:

$$\begin{cases} Ay \ge f; \\ y \ge 0;\ y \le 1; \\ b^{\mathrm{T}} y \to \max. \end{cases}$$

Здесь искомый вектор $y = [\{\pi_i\}\{\gamma_{ij}\}]$ имеет размерность, равную сумме числа вершин и числа ветвей графа сети плюс единица: $|y| = |\pi_i| + |\gamma_{ij}| + 1 = 5 + 7 + 1 = 13$. Матрица ограничений $A = \left[A_{eq}^{\mathrm{T}} \,\vdots\, A_{ineq}^{\mathrm{T}} \right]$ составляется из транспонированных матриц систем равенств и неравенств прямой задачи (см. пример 1.2) и имеет размеры (число ветвей +1) × (число элементов вектора *y*), т. е. 8×13:

		A_{eq}^{T}						A_{ineq}^{T}				
p_1	p_2	p_3	p_4	p_5	t_{12}	t_{13}	t_{14}	t_{43}	t_{32}	t_{25}	t_{45}	(Ф)
−1	1	0	0	0	1	0	0	0	0	0	0	0
−1	0	1	0	0	0	1	0	0	0	0	0	0
−1	0	0	1	0	0	0	1	0	0	0	0	0
0	0	1	−1	0	0	0	0	1	0	0	0	0
0	1	−1	0	0	0	0	0	0	1	0	0	0
0	−1	0	0	1	0	0	0	0	0	1	0	0
0	0	0	−1	1	0	0	0	0	0	0	1	0
1	0	0	0	−1	0	0	0	0	0	0	0	−1

В составе вектора *f* двойственной задачи, как и в линейной форме прямой задачи, восемь элементов, семь из которых относятся к ветвям графа и равны нулю, а восьмой относится к переменной Ф и равен −1:

$$f^{\mathrm{T}} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ -1].$$

Такое содержание вектора *f* означает, что в двойственной задаче ищется максимум только одного граничного потока, притом через его минимум (коэффициент при Ф равен −1).

Вектор b линейной формы имеет такую же длину (13 элементов), как и искомый вектор y, и так же разбит на три части, относящиеся к вершинам, ветвям и потоку минимального сечения (табл. 1.2).

Таблица 1.2

Состав и значения векторов b и y двойственной задачи

	Вершины					Ветви							Поток
Вектор	p_1	p_2	p_3	p_4	p_5	t_{12}	t_{13}	t_{14}	t_{43}	t_{32}	t_{25}	t_{45}	(Ф)
b	0	0	0	0	0	3	8	4	6	4	10	5	1
y	π_1	π_2	π_3	π_4	π_5	γ_{12}	γ_{13}	γ_{14}	γ_{43}	γ_{32}	γ_{25}	γ_{45}	–

Ограничения двойственной задачи содержат нижние — нулевые и верхние — единичные границы для всех элементов искомого вектора y. Таким образом ее решение — вектор y — ограничивается поиском действительно минимального сечения.

1.2.4. Решение задачи определения минимального сечения

Для того чтобы получить решение двойственной задачи, следует использовать готовый программный инструмент — систему моделирования MATLAB.

M-функция решения оптимизационной задачи линейного программирования (1.1) называется **linprog**. Она имеет следующий формат обращения [5]:

```
[x, fval, exitflag, output, lambda] = …
linprog (f, A, b, Aeq, beq, lb, ub, x0, options)
```

Входные параметры команды:

A и **Aeq** — матрицы, определяющие систему неравенств (**A*x <= b**) и равенств (**Aeq*x = beq**) в задаче линейного программирования соответственно. При отсутствии равенств на место **Aeq** ставится [];

f — вектор, задающий условие оптимального решения;

b и **beq** — векторы правых частей системы неравенств и равенств соответственно. При отсутствии равенств на место **beq** ставится пустой массив [];

lb и **ub** — векторы нижних и верхних границ интервалов, внутри которых должен находиться искомый вектор (**lb <= x <= ub**);

x0 — вектор начального приближения. Используется только в режиме обычного поиска (**medium-scale algorithm**) оптимального решения;

options — структура, содержащая значения свойств алгоритма оптимизации. Для определения этих свойств следует обращаться к *M*-функции **optimset**, например, задавать относительную погрешность определения оптимального решения в виде **optimset ('tol',1e-6)**.

Выходные параметры команды:

x — искомый вектор оптимального решения;

fval — значение (скаляр) оптимального решения (**fval = f'*x**);

exitflag — числовая константа, определяющая условия окончания поиска оптимального решения;

output — структура, содержащая информацию об оптимальном решении;

lambda — структура, поля которой содержат множители Лагранжа при определении оптимального решения.

Пример 1.4. Решение двойственной задачи линейного программирования для транспортной сети в системе моделирования MATLAB.

Подставим конкретные входные данные для решения двойственной задачи в *m*-функцию **linprog**, определенные в примере 1.3. Принятые в *m*-функции обозначения оставлены без изменения, дополнительно заданы только границы **lb** и **ub** значений элементов искомого вектора **y**.

```
A = [-1 1 0 0 0 1 0 0 0 0 0 0 0;…   % Приведена частично
     -1 0 1 0 0 0 1 0 0 0 0 0 0;…
..............
1 0 0 0 -1 0 0 0 0 0 0 0 -1];
f = [0; 0; 0; 0; 0; 0; 0; -1];
b = [0; 0; 0; 0; 0; 3; 8; 4; 6; 4; 10; 5; 1];
lb = zeros (size (A,2),1); % Нижняя граница
ub = ones (size (A,2),1);  % Верхняя граница
[y, fval] = …
linprog (f,– A, b, [], [], lb, ub, [], optimset ('Display','iter'))
```

Здесь знак матрицы ограничений изменен на обратный, поскольку стандартная операция отношения в *m*-функции есть «меньше или равно», а двойственная система требует другой операции — «больше или равно». Вектор нижних границ — нулевой, вектор верхних границ — единичный. Длина обоих векторов равна длине вектора **y**, т. е. 13.

После 8 итераций искомое решение в командном окне примет вид:

```
Residuals: Primal   Dual        Upper      Duality    Total
           Infeas   Infeas      Bounds     Gap        Rel
           A*x-b    A'*y+z-w-f  {x}+s-ub   x'*z+s'*w  Error
-----------------------------------------------------------
Iter   0: 2.01e+002  1.96e+001  7.18e+002  2.77e+004  2.01e+002
Iter   1: 2.10e+001  9.19e-015  7.51e+001  3.27e+003  2.10e+001
Iter   2: 1.09e+000  2.08e-013  3.90e+000  2.73e+002  1.61e+000
Iter   3: 2.22e-001  1.05e-012  7.92e-001  9.35e+001  1.65e+000
Iter   4: 2.39e-002  1.90e-012  8.54e-002  1.50e+001  8.21e-001
Iter   5: 2.07e-003  4.66e-013  7.37e-003  3.27e+000  2.51e-001
Iter   6: 1.18e-004  3.38e-012  4.22e-004  2.58e-001  2.26e-002
Iter   7: 2.48e-007  1.37e-014  8.85e-007  4.83e-004  4.27e-005
Iter   8: 1.24e-011  3.67e-014  4.42e-011  2.42e-008  2.14e-009
Optimization terminated.
y = 1.0 0.0 1.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0
fval = 11.0000
```

При детальном анализе решения оказалось, что единичные значения вектора **y** действительно представляют собой целые числа, в то время как его «нулевые» значения вычислены приближенно (табл. 1.3). Индивидуальная погрешность вычислений элементов вектора **y** сравнима с относительной погрешностью, достигнутой на восьмой итерации, т. е. около 2 e-9 (в примере максимальная погрешность равна 1,21e-9).

Таблица 1.3

Параметры минимального сечения транспортной задачи

	Вершины					Ветви							Поток
Вектор	π_1	π_2	π_3	π_4	π_5	γ_{12}	γ_{13}	γ_{14}	γ_{43}	γ_{32}	γ_{25}	γ_{45}	Ф
y	1	0	1	0	0	1	0	1	0	1	0	0	11

На рис. 1.7 представлена копия рабочего пространства системы MATLAB при решении двойственной задачи и графическое изображение анализируемой транспортной сети с выделенным минимальным сечением. Вершины графа, относящиеся к истоку, помечены.

а *б*

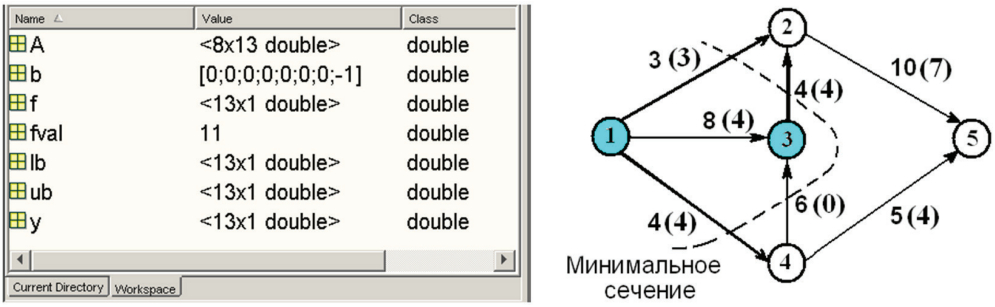


Рис. 1.7. Содержание рабочего пространства при поиске минимального сечения — *а* и граф транспортной сети с найденным минимальным сечением — *б*

1.2.5. Автоматизированный поиск максимального потока в сети

Приведенные выше примеры решения прямой и двойственной задачи носят весьма иллюстративный характер и вряд ли могут быть напрямую использованы на практике. Ручное составление матриц и векторов для большеразмерных транспортных сетей, как правило, сопровождается ошибками (описками или неучетом некоторых данных), которые затруднительно отыскать и исправить. Очевидно, что большую часть рутинной работы по подготовке входной информации для *m*-функции `linprog` может выполнить сама ЭВМ с помощью специальной программы. Покажем, как это сделать в системе моделирования MATLAB.

Обозначим фрагменты общей задачи поиска максимального потока.

1) Графический ввод данных о графе транспортного потока.
2) Получение текстовой информации о топологии графа.
3) Формирование списочной модели транспортного потока.
4) Формирование содержания всей входной информации для `linprog`.
5) Решение прямой задачи линейного программирования.
6) Обработка полученных результатов и выработка рекомендаций.

Перечисленные выше подзадачи могут быть автоматизированы в различной степени. Например, первая подзадача требует разработки специального графического интерфейса, с помощью которого на экране монитора можно отобразить в различных масштабах всю сеть или любой ее фрагмент. Примеры подобных интерфейсов широ-

ко используются при графическом вводе принципиальных электрических схем в программах схемотехнического моделирования. Один из них — графический редактор программы Micro-Cap — представлен на рис. 1.8, *в*, где в виде направленных ветвей графа (рис. 1.8, *а*) использованы графические образы индуктивности. У этого элемента имеются разноименные выводы, что позволяет использовать его ориентацию: положительным считается направление (тока) от плюса к минусу. Хотя имеется возможность переименования узлов такого квазиграфа, применение подобной «автоматизации» для формирования списочной модели (рис. 1.8, *б*) даже средней по размерам транспортной сети вряд ли возможно. В этом случае необходимо обращаться к специализированным графическим инструментам ввода и обработки топологической информации, созданными для работы с ориентированными графами.

а *б* *в*

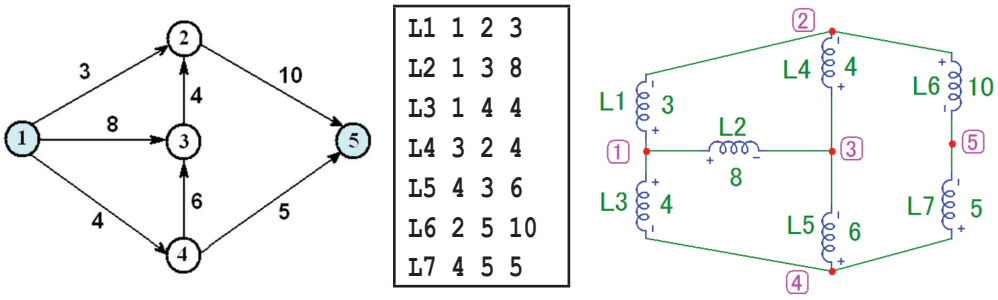


Рис. 1.8. Граф транспортной сети — *а*, его списочная модель — *б* и пример его представления в среде программы Micro-Cap — *в*

Все другие подзадачи (со второй по шестую), названные в начале параграфа, в той или иной степени могут быть автоматизированы. При отсутствии графического интерфейса формирование списочной модели транспортной сети необходимо выполнить вручную в любом текстовом редакторе. Ввод данных следует проводить по следующим правилам:

1) перед началом ввода иметь рисунок транспортной сети с числовыми обозначениями вершин: первая — исток, последняя — сток;
2) для каждой ветви указывается ее пропускная способность;
3) каждая строка списочной модели соответствует одной направленной ветви и представляет данные о ней в виде четырех чисел:

<Номер ветви по порядку> <Номер начала> <Номер конца> <Пропускная способность>

4) результаты ввода данных о сети записываются в текстовый файл с расширением `*.lst`, например, `D:\TransNet\tn.lst`.

Заметим, что ввод номера ветви для последующих расчетов не нужен, однако он весьма полезен при отработке списков из многих десятков и сотен строк. Кроме того, первые три числа имеют целый тип, четвертое число — в общем случае вещественный тип.

Ниже приведен текст скрипт-файла `TransNet`, позволяющего получить численное решение поставленной в начале параграфа задачи.

```
% ======= Программа расчета максимального потока ========
fname = 'tn.lst';           % Поиск файла в стандартном окне Open
filename = uigetfile({'*.lst','All *.lst files';...
          '*.*','All files' }, 'Open file', fname);
fid = fopen(filename,'r');    % Указатель найденного файла
PT = [];                    % Массив для хранения данных
l=1;
while ~feof(fid)            % Цикл по чтению строк
    tline = fgetl(fid);
    if ischar(tline) && ~isempty(tline)
        PT(l,:) = str2num(tline);  % Преобразование в числа
        l=l+1;
    end
end
c = PT(:,4);    % Ввод массива параметров ветвей
pt = PT(:,2:3); % Выделение номеров узлов у ветвей
n = max(pt(:)); % Определение максимального номера узла в графе
m = size(pt,1); % Определение числа ветвей в графе
A = zeros(n,m); % Нулевой шаблон матрицы связей в графе

for i=1:m               % Заполнение матрицы связей
    A(pt(i,1),i)=-1;    % (или инциденций)
    A(pt(i,2),i)=+1;    % (Входные и выходные узлы ветвей)
end

Aeq = [A [1 zeros(1,n-2) -1]']; % Дополнение матрицы связей
% вектором, определяющим равенство входного и выходного потоков
beq = zeros(n,1);       % Нулевой вектор правых частей равенств
```

```
Aineq = eye(m+1);          % Единичная матрица неравенств
Aineq(end,end) = -Aineq(end,end); % Ограничение выходного потока
bineq = [c;0];             % Границы в виде пропускных способностей
f = [zeros(m,1); -1];      % Определение линейной формы

lb = zeros(m+1,1);         % Нижние границы значений потоков
ub = [c;Inf];              % Верхние границы значений потоков

% Численное решение транспортной задачи
[x, fval] = linprog(f, Aineq, bineq, Aeq, beq, lb,ub,[]);
[' fval = ' num2str(-fval)]
cx = [pt  c  x(1:end-1,1)]  % Вывод таблицы на экран
```

Ниже в примере показано применение скрипт-файла для анализа сети.

Пример 1.5. Компьютерное моделирование потоков в транспортной сети.

Рассмотрим транспортную сеть, ориентированный граф которой изображен на рис. 1.9. Граф имеет 12 вершин, из которых первая (исток) и двенадцатая (сток) вершины — конечные, и 25 ветвей с данными о пропускных способностях. Граф содержит полную информацию о транспортной сети. Отметим, что сумма пропускных способностей конечных ветвей со стороны стока и истока одинакова и равна 60.

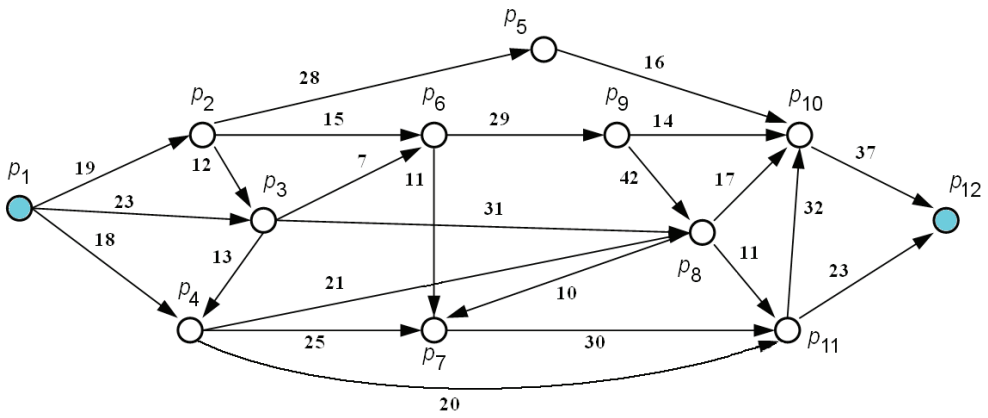


Рис. 1.9. Граф транспортной сети

Определим три задачи исследования этой сети:

1) поиск максимального потока и ветвей, входящих в критическое сечение для заданной транспортной сети;
2) решение такой же задачи для случая, когда ветви графа, исходящие из истока и входящие в сток, имеют неограниченную пропускную способность. Эта задача означает поиск действительно максимальной пропускной способности транспортной сети;
3) решение первой задачи для несбалансированной транспортной сети.

Решение этих задач проведем с помощью системы моделирования MATLAB, для чего сформируем двумерный массив `PT` параметров ветвей. В нем четыре столбца и число строк, равное числу ветвей графа. Первый столбец хранит номера ветвей, далее два столбца содержат начальный и конечный вершины каждой ветви, в четвертом столбце указывается ее пропускная способность. Этот массив должен быть определен в рабочем пространстве (Workspace). Вторая задача решается после первой, тогда пропускные способности конечных ветвей, сохраненные в рабочем пространстве, можно изменить в командном окне.

В результате решения *первой задачи* получено распределение общего потока по сети в размере 60 единиц, причем ни одна из ветвей, кроме конечных, не «загружена» полностью, т. е. все внутренние ветви сети имеют величину потока меньше их пропускной способности. Значения потоков каждой ветви представлены в четвертом столбце табл. 1.4.

Таблица 1.4

Потоки в исследуемой транспортной сети

Номер начала	Номер конца	Размер пропуска	Поток	
			Задача 1	Задача 2
1	2	19	19	32,404
1	3	23	23	29,810
1	4	18	18	53,787
2	5	28	9,0456	16
2	6	15	8,3876	15
2	3	12	1,5667	1,4037
3	6	7	3,3732	7
3	8	31	16,163	21,718
3	4	13	5,0307	2,4955
4	7	25	10,852	24,168
4	8	21	2,9514	12,114
4	11	20	9,2275	20
5	10	16	9,0456	16
6	7	11	0,6911	0
6	9	29	11,070	22
7	11	30	15,336	30
8	7	10	3,7933	5,8321
8	10	17	10,751	17
8	11	11	4,5697	11
9	10	14	1,5100	2,7486
9	11	42	9,5597	19,251
10	12	37	37	48,137
11	12	23	23	67,863
11	10	32	15,693	12,389
			60	116

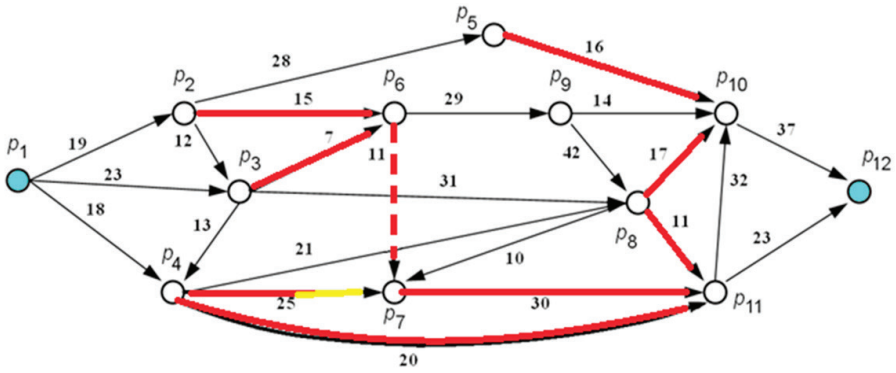
Итак, максимальный поток в исследуемой транспортной сети (60 единиц) ограничен пропускными способностями конечных ветвей.

Численное решения *второй задачи* гораздо интереснее. Во-первых, при снятии ограничений на граничные ветви почти в два раза возрос

общий поток в сети — с 60 единиц до 116 единиц. Во-вторых, определены ветви графа, относящиеся к критическому сечению: их восемь, они помечены стрелками в пятом столбце табл. 1.4, содержащем значения потоков ветвей во второй задаче. Ветвь t_{6-7} имеет нулевой поток и в такой сети лишняя. Ветвь t_{4-7} близка, но не входит в критическое сечение (поток 24,168 единиц вместо 25 единиц).

Граф транспортной сети с выделенным критическим сечением показан на рис. 1.10, *а*, дополнение критического сечения — на рис. 1.10, *б*. Заметим, что критическое сечение не разрывает исходный граф, поскольку дополнение все-таки остается односвязным. Подобная «витиеватость» критического сечения не очевидна даже в такой простой сети, она может быть обнаружена только в результате расчета всех потоков в транспортной сети.

а



б

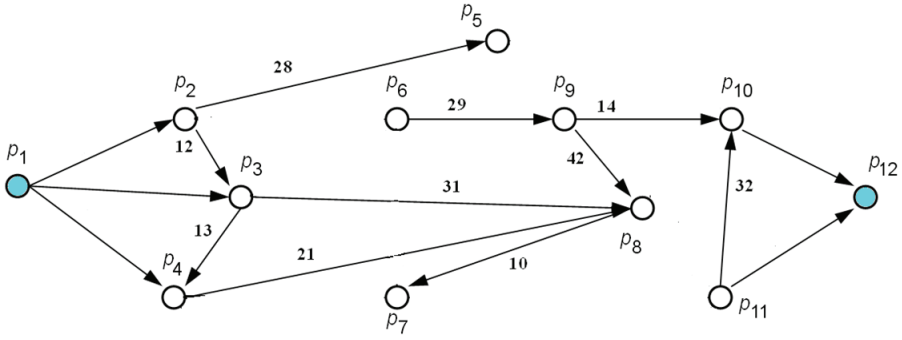


Рис. 1.10. Граф транспортной сети с выделенными ветвями критического сечения — *а* и дополнение критического сечения — *б*

Третья задача — анализ несбалансированной транспортной сети — решается на основе метода, предложенного в конце параграфа 1.1

(см. рис. 1.4). Рассмотрим сначала вариант превышения пропускной способности стока ($\Gamma_1 > \Gamma_N$). Оставим пропускную способность стока $\Gamma_1 = 60$ без изменения, а пропускную способность стока уменьшим на 20 единиц ($\Gamma_N = 40$). Добавим в конец списочной модели исходной транспортной сети три строки (три ветви и две вершины на графе на рис. 1.4, *а*). Вершины пронумеруем в нарастающем порядке от конечной ранее с номером 12, т. е. 13 и 14. Первой ветви t_{1-13} от начальной вершины 1 до предконечной с номером 13 присвоим «бесконечную» пропускную способность 1000. Второй ветви t_{12-14} от бывшей конечной вершины с номером 12 до новой конечной с номером 14 присвоим «сниженную» пропускную способность $\Gamma_N = 40$. Третьей ветви t_{13-14} от предконечной вершины с номером 13 до конечной с номером 14 присвоим разность пропускных способностей $\Gamma_1 - \Gamma_N = 20$.

В этом случае списочная модель сбалансированной сети и результаты ее численного решения представлены в левом столбце табл. 1.5.

Таблица 1.5

Списочные модели и потоки в сбалансируемых транспортных сетях

НУ	КУ	Инт	Поток	НУ	КУ	Инт	Поток
1	2	19	13.215	13	2	19	19
1	3	23	15.361	13	3	23	23
1	4	18	11.424	13	4	8	8
2	5	28	6.2748	2	5	28	8.5302
2	6	15	5.6501	2	6	15	8.5024
2	3	12	1.2902	2	3	12	1.9674
3	6	7	2.6121	3	6	7	3.1238
3	8	31	10.9470	3	8	31	15.321
3	4	13	3.0925	3	4	13	6.5229
4	7	25	6.0414	4	7	25	7.2898
4	8	21	1.9706	4	8	21	1.8122
4	11	20	6.5044	4	11	20	5.4209
5	10	16	6.2748	5	10	16	8.5302
6	7	11	1.4176	6	7	11	0.90717
6	9	29	6.8446	6	9	29	10.719
7	11	30	9.6928	7	11	30	12.245
8	7	10	2.2338	8	7	10	4.0482
8	10	17	7.4677	8	10	17	9.0008
8	11	11	3.2157	8	11	11	4.0839
9	10	14	2.0019	9	10	14	1.3414
9	11	42	4.8427	9	11	42	9.3776
10	12	37	25.493	10	12	37	31.51
11	12	23	14.507	11	12	23	18.49
11	10	32	9.7483	11	10	32	12.637
1	13	1000	20	1	13	50	50
12	14	40	40	1	14	10	10
13	14	20	20	12	14	1000	50

Очевидно, что суммарный поток истока по трем «старым» ветвям, равный 13,215 + 15,361 + 11, 424 = 40 единиц, автоматически устанавливается равным величине пропускной способности на выходе (40 = 60–20).

Вариант превышения пропускной способности истока ($\Gamma_N > \Gamma_1$ или 60 > 50) формализуется (см. рис. 1.4, *б*) и решается аналогичным образом. Небольшое отличие состоит лишь в том, что обе прежние конечные вершины перенумеруются: начальная в предконечную (1 → 13), конечная в новую конечную (12 → 14). Списочная модель сбалансированной таким образом сети и результаты ее численного решения показаны в правом столбце табл. 1.5.

1.3. Модели на основе максимального потока

Существует ряд задач, математические модели которых близки, а графическое представление похоже на графы транспортных сетей. Эти задачи решаются либо с помощью специального алгоритма, названного по работам Куна венгерским [1, 8, 9], либо преобразуются в задачи о максимальном потоке с последующим решением численными методами линейного программирования.

Формулировка таких задач в терминах теории графов приводит к двудольным графам, их обобщенное структурное изображение представлено на рис. 1.11 в виде внутреннего содержания штрихового прямоугольника. Двудольный граф имеет n вершин с одной стороны и m вершин с другой стороны, причем каждая вершина одной (левой) стороны соединена ветвями со всеми вершинами другой (правой) стороны. В общем случае таких ветвей должно быть $n \times m$, однако некоторые из них могут отсутствовать. Если ветви ориентированы, как, например, на рис. 1.11, то такой граф называется ориентированным двудольным графом.

Графическая формулировка обсуждаемых ниже задач требует дополнения двудольного графа двумя конечными вершинами, как это было в случае транспортных сетей: первая обозначается x_0 и называется истоком, вторая обозначается y_0 и называется стоком. Обе конечные вершины соединены со всеми соответствующими вершинами направленными ветвями, и согласно свойству транспортных сетей поток,

исходящий из конечной вершины x_0, равен потоку, входящему в конечную вершину y_0. Заметим, что после введения стока и истока указание на ориентацию ветвей особого смысла не имеет.

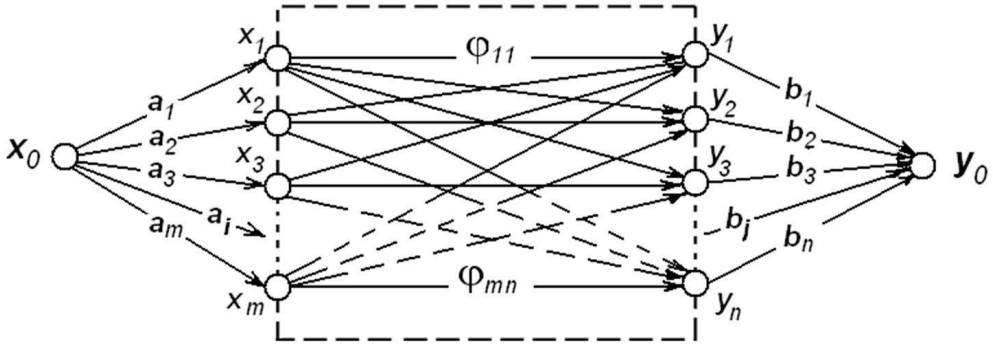


Рис. 1.11. Граф задач о распределении

Рассмотрим несколько задач, имеющих такой граф.

1.3.1. Задачи оптимального распределения

Пример 1.6. Транспортная задача распределения Хичкока [1].

Задано m портов отправления x_i и n портов назначения y_j, количество судов a_i, которое должно быть отправлено из порта x_i, количество судов b_j, которое должно прийти в порт y_j, и транспортные расходы d_{ij} при перевозке грузов из портов x_i в y_j. Необходимо определить количество судов φ_{ij}, которые следует отправить из порта x_i в порт y_j, чтобы общие расходы на перевозки были минимальные, т. е. чтобы

$$L = \sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij}\varphi_{ij} \to \min. \tag{1.28}$$

Требуется найти целочисленные положительные значения d_{ij}, которые помимо условия (1.18) удовлетворяют ограничениям

$$\begin{cases} \sum\limits_{i=1}^{m} d_{ij} = b_j, \quad j = 1,2,\ldots,n; \\ \sum\limits_{j=1}^{n} d_{ij} = a_i, \quad i = 1,2,\ldots,m. \end{cases} \tag{1.29}$$

Имеется другая формулировка задачи Хичкока, где равенства (1.29) и так называемое условие существования (условие сохранения потока от истока к стоку, т. е. условие сбалансированности задачи по потоку)

$$\sum_{i=1}^{n} b_j = \sum_{i=1}^{m} a_i \tag{1.30}$$

заменяются неравенствами

$$\begin{cases} \sum\limits_{j=1}^{n} d_{ij} \le a_i; \\ \sum\limits_{i=1}^{m} d_{ij} \ge b_j; \\ \sum\limits_{i=1}^{m} a_i \ge \sum\limits_{j=1}^{n} b_j \ . \end{cases} \tag{1.31}$$

Несбалансированную задачу в форме неравенств можно свести к равенствам, т. е. к сбалансированной задаче о распределениях, если к матрице $[d_{ij}]$ добавить еще один $(n+1)$-й столбец с суммой элементов, равной разности $\sum\limits_{i=1}^{m} a_i - \sum\limits_{j=1}^{n} b_j$ и положить $d_{i,n+1} = 0$. При этом существенно, чтобы все d_{ij} были неотрицательны.

Пример 1.7. Транспортная задача Ордена [1].

Имеется излишек a_i товара в m пунктах $x_i, i = 1,2,\ldots,m$, и спрос b_i в n пунктах y_j, $j = m+1, m+2,\ldots,n+m$. Заданы пропускные способности c_{kl}, $k,l = 1,2,\ldots,n+m$, между пунктами и транспортные расходы d_{kl} по перевозке единицы товара из одного пункта в другой. Требуется определить оптимальные значения φ_{ij} количества товара, которые следует перевозить, чтобы суммарные транспортные расходы были минимальны, т. е. чтобы

$$L = \sum_{k} \sum_{l} d_{kl} \varphi_{kl} \to \min \tag{1.32}$$

при условии равенства общего излишка товара и общего его спроса

$$\sum_{j=m+1}^{m+n} b_j = \sum_{i=1}^{m} a_i. \tag{1.33}$$

Задача решается при следующих ограничениях:

$$\begin{cases} \sum\limits_{i=1}^{m} \varphi_{ii} = b_i, \quad j = m+1,\ m+2, \ldots, m+n; \\ \sum\limits_{j=m+1}^{m+n} \varphi_{ij} = a_i, \quad i = 1, 2, \ldots, m; \\ 0 \le \varphi_{kl} \le c_{kl}. \end{cases} \tag{1.34}$$

Пример 1.8. Задача об оптимальном распределении задач ЭВМ. Требуется распределить набор задач $x_i, i = 1,2,\ldots,n$, по разным ЭВМ $y_j, j = 1,2,\ldots,n$. Задана некоторая положительная величина d_{ij}, характеризующая качество решения задачи x_i на ЭВМ y_j. Условимся, что $\varphi_{ij} = 1$, если задача x_i распределена на ЭВМ y_j, и $\varphi_{ij} = 0$, если задача не назначается на эту ЭВМ. Требуется максимизировать функцию

$$D = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} \varphi_{ij} \to \max, \tag{1.35}$$

т. е. найти такое оптимальное распределение задач по ЭВМ, которое обеспечивает максимальную суммарную характеристику при ограничениях

$$\begin{cases} \sum\limits_{i=1}^{n} \varphi_{ij} = 1, \ j = 1,2,\ldots,n; \\ \sum\limits_{j=1}^{n} \varphi_{ij} = 1, \ i = 1,2,\ldots,n. \end{cases} \tag{1.36}$$

Ограничение (1.36) означает, что исключается распределение нескольких задач на одну ЭВМ и обработку несколькими ЭВМ одной задачи.

Если в задаче о распределении не соблюдается равенство числа задач n и числа исполнителей, например, $n>m$, то ко входу добавляется $k = n - m$ фиктивных задач с суммарной пропускной способностью $\Delta\Gamma = \sum\limits_{j=1}^{n} b_j - \sum\limits_{i=1}^{m} a_i$ и с «бесконечной» стоимостью выполнения: $\varphi_{ij} \to \infty,\ i = (n+1)..(n+k),\ j = 1..m$. Аналогичная процедура «выравнивания» числа задач и исполнителей проводится при обратном дисбалансе, когда $n<m$. В любом случае величина разности $\Delta\Gamma$ должна быть неотрицательной.

В задачах сетевого планирования и управления часто встречается задача определения критического пути. Любая сложная комплексная работа или разработка изображается в виде сетевого графика, который представляет собой некоторую транспортную сеть. Начальная точка x_0 этой сети соответствует началу работ по комплексу, конечная точка y_0 — окончанию. Каждая отдельная (частная) работа комплекса представляется в виде ветви, начальная вершина которой соответствует началу работы, а конечная вершина — концу работы; каждой ветви (x_i, y_i) ставится в соответствие число t_{ij} — время выполнения работы, каждой вершине — время $T_i, i = \overline{1,m}; \quad T_j, j = \overline{1,n}$, наступления данного события (начала работы).

Если ввести в рассмотрение еще и стоимость c_{ij} выполнения работ, то в данном критическом пути следует сокращать те работы, которые обладают наименьшим увеличением стоимости при сокращении времени выполнения работы. Если обозначить зависимость стоимости выполнения работы от времени через $c_{ij} = f_{ij}(t_{ij}) \geq 0$, то сокращать следует ту работу критического пути, для которой производная $\dfrac{dc_{ij}}{dt_{ij}} = \dfrac{df_{ij}(t_{ij})}{dt_{ij}}$ максимальна. После сокращения времени одного критического пути может появиться другой критический путь. Далее, если ввести моменты выполнения каждой работы (T_i, T_j), то появится возможность уменьшить стоимость выполнения комплекса работ за счет уменьшения стоимости отдельных работ, не лежащих на критическом пути (но имеющих резерв времени) путем увеличения времени их выполнения.

Данную задачу можно свести к отысканию потока минимальной стоимости, если вместо времени выполнения работ t_{ij} ввести величины $l_{ij} = h - t_{ij}$, где $h = \max\limits_{i,j} t_{ij}$, или рассматривать отрицательные величины $\tilde{t}_{ij} = -t_{ij}$, что не вносит принципиальных затруднений.

1.3.2. Математическая модель оптимального распределения

Стоит заметить, что частная задача о паросочетании может быть решена более эффективно, поскольку алгоритм нахождения максимального потока является общим и не использует двудольную природу исходной сети.

Сведем рассмотренные три задачи к единой математической формулировке. Будем считать, что во всех задачах требуется минимизировать функцию

$$L(\varphi) = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\varphi_{ij} \tag{1.37}$$

при ограничениях

$$\begin{cases} \sum\limits_{j=1}^{n} \varphi_{ij} = a_i, \quad i = 1,2,\ldots,n; \\ \sum\limits_{i=1}^{n} \varphi_{ij} = b_j, \quad j = 1,2,\ldots,n; \\ 0 \le \varphi_{ij} \le c_{ij}. \end{cases} \tag{1.38}$$

Задачу максимизации можно свести в общем случае к задаче минимизации одним из двух способов. Во-первых, можно вместо переменных d_{ij} ввести величины

$$l_{ij} = h - d_{ij}, \tag{1.39}$$

где

$$h \ge \max d_{ij}. \tag{1.40}$$

Если рассмотреть тождество

$$\sum_{i=1}^{n}\sum_{j=1}^{n} l_{ij}\varphi_{ij} = h\sum_{i=1}^{n} a_i - \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}\varphi_{ij}, \tag{1.41}$$

то максимизация выражения вида (1.39) эквивалентна минимизации правой части тождества (1.41). Второй способ состоит в замене положительных значений φ_{ij} их отрицательными значениями.

В ряде случаев (задача о минимальном пути) отрицательные значения потока не служат препятствием к решению задачи, однако в общем случае для рассмотренных трех задач предпочтительнее выбрать первый способ.

Задача Хичкока сводится к стандартной формулировке, если положить $c_{ij} = \infty$. Как будет видно из дальнейшего, величины c_{ij} являются пропускными способностями ветвей.

Для задачи об оптимальном назначении необходимо вместо величин d_{ij} ввести l_{ij} в соответствии с формулой (1.39) и в формулах (1.37) и (1.38) положить

$$a_i = 1,\ b_j = 1,\ c_{ij} = 1,\ i, j = 1, 2, \ldots, n. \tag{1.42}$$

Это означает, что φ_{ij} может принимать только два значения — либо 0, либо 1 — в силу условия целочисленности потока φ_{ij}.

Стандартную задачу можно представить в виде графа, изображенного на рис. 1.11. Вершины x_i, $i = 1, 2, \ldots, m$, соответствуют пунктам отправления или задачам, а вершины y_j, $j = 1, 2, \ldots, n$, — пунктам назначения или ЭВМ. В графе вершина-исток x_0 соединена с вершинами x_i ветвями с пропускной способностью a_i, вершина-сток y_0 соединена с вершинами y_j ветвями с пропускной способностью a_j, где

$$\begin{cases} \sum\limits_{j=1}^{n} \varphi_{ij} = \varphi_{0i}\ ; \\ \sum\limits_{i=1}^{m} \varphi_{ij} = \varphi_{j0}\ ; \\ c_{0i} = a_i\ ; \\ c_{j0} = b_j\ . \end{cases} \tag{1.43}$$

Такой граф является графической моделью приведенных выше задач. Заметим, что в задаче о назначениях никаких транспортных путей нет, а модель имеет вид графа транспортной сети. Это изображение отражает топологию задачи, так же как структурные схемы систем автоматического регулирования отражают в геометрическом образе внутреннюю структуру системы управления.

Пример 1.9. Решение задач оптимального распределения.

Два одинаковых коммутационных пункта — пункт А и пункт В — имеют по восемь входных и выходных портов, которые соединены между собой 64 каналами передачи данных. Интенсивность передачи данных через выходные порты пункта А представлена вектором

$$\mathbf{a} = [12, 12, 8, 8, 5, 5, 15, 15]^{\mathrm{T}}\ \text{Мбит/с},$$

а через входные порты коммутационного пункта В — вектором

$$\mathbf{b} = [16, 4, 16, 4, 16, 4, 16, 4]^{\mathrm{T}}\ \text{Мбит/с}.$$

Стоимость передачи данных по каждому из 64 каналов задается матрицей

$$d = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} \mathbf{22.49} & 41.99 & 42.93 & 68.34 & 56.74 & 44.51 & 34.87 & 51.26 \\ 23.92 & 37.00 & 63.49 & 53.24 & 54.36 & 60.79 & \mathbf{22.46} & 56.67 \\ 52.04 & 35.71 & 66.71 & 63.52 & 37.30 & 43.04 & 54.66 & 38.79 \\ 29.54 & 38.25 & 33.22 & \mathbf{20.49} & 28.30 & 42.87 & 52.50 & \mathbf{20.49} \\ 62.19 & 39.66 & \mathbf{28.01} & 26.85 & \mathbf{27.78} & 42.53 & 69.15 & 40.99 \\ 28.69 & 49.57 & 63.64 & 60.94 & 29.55 & 40.61 & 47.63 & 57.68 \\ 28.54 & 25.98 & 31.89 & 41.51 & 41.12 & \mathbf{6}5.08 & 40.00 & 59.69 \\ 69.71 & \mathbf{21.90} & 52.29 & 64.51 & 62.80 & \mathbf{20.28} & 29.94 & 66.00 \end{bmatrix} \end{array} \text{ руб/мин.}$$

В матрице *d* полужирным шрифтом выделены минимальные по каждому столбцу элементы.

Необходимо определить номера каналов φ_{ij}, которые следует использовать для передачи данных от пункта А в пункт В, чтобы общие расходы на передачу информации были минимальные, т. е. чтобы

$$L = \sum_{i=1}^{8}\sum_{j=1}^{8} d_{ij}\varphi_{ij} \to \min.$$

Требуется найти целочисленные положительные значения φ_{ij}, которые удовлетворяют ограничениям в виде равенств:

$$\sum_{j=1}^{8} b_j\varphi_{ij} = \sum_{i=1}^{8} a_j\varphi_{ij}.$$

1. Сначала рассмотрим вариант одинаковой интенсивности передачи данных через коммутационные порты, приняв их равными $\sum_{i=1}^{8} a_i / 8 = \sum_{j=1}^{8} b_i / 8 = 10$ Мбит/с. В этом случае решение оптимизационной задачи сводится к выбору в матрице *d* восьми чисел, расположенных обязательно по одному в каждой строке и каждом столбце, причем сумма этих чисел должна быть минимальной. Обратимся к *m*-функции решения оптимизационной задачи линейного программирования в системе MATLAB

```
[x, fval] = linprog (f, [], [], A, b, lb, ub, [],...
                optimset ('Display','iter')),
```

где `f = d(:);` — вектор, задающий условие оптимального решения, который составлен из столбцов матрицы *d*. Его длина равна количеству элементов матрицы, т. е. 64;

A — матрица системы равенств, которая формируется из двух матриц: матрицы **A1** — при умножении на искомый вектор служит для суммирования по строкам, матрицы **A2** — по столбцам. Матрица **A** содержит 8+8=16 строк и 64 столбца.

```
A1 = [[ones (1,8); zeros (7,8)]
[zeros (1,8); ones (1,8); zeros (6,8)]…
[zeros (2,8); ones (1,8); zeros (5,8)]…
[zeros (3,8); ones (1,8); zeros (4,8)]…
[zeros (4,8); ones (1,8); zeros (3,8)]…
[zeros (5,8); ones (1,8); zeros (2,8)]…
[zeros (6,8); ones (1,8); zeros (1,8)]…
[zeros (7,8); ones (1,8)]];
A2 = repmat (eye (8),1,8);
A = [A1; A2];
```

b = ones(16,1); — вектор правых частей системы равенств;

lb = zeros(64,1); — вектор нижних и

ub = ones(64,1); — вектор верхних границ интервалов, внутри которых должен находиться искомый вектор (**lb <= x <= ub**);

[] — обозначения отсутствующих во входном списке матриц и векторов (здесь это относится к массивам, определяющим систему неравенств).

В результате решения был получен целочисленный (с погрешностями вычислений порядка 1 e-14) вектор **x**, состоящий из 64 элементов, среди которых было 8 единиц и 56 нулей. Местоположение этих единиц соответствовало выбранным каналам. Матрица стоимостей передачи данных выбранных каналов имеет вид:

$$d_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} \mathbf{22.49} & 0 & \mathbf{0} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{22.46} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 38.79 \\ 0 & 0 & 0 & \mathbf{20.49} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{28.01} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 29.55 & 0 & 0 & 0 \\ \mathbf{0} & 25.98 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{20.28} & 0 & 0 \end{bmatrix} \end{array} \text{ руб/мин.}$$

В матрице d_1 полужирным шрифтом выделены элементы, которые совпадают с минимальными значениями по столбцу в исходной матрице d.

Номера выходных и входных коммутационных портов определялись с помощью двух команд:

```
fi = reshape(x,8,8);
fij = sparse(fi > 1e-6)
```

Минимальная стомость передачи данных по всем выбранным каналам составила `fval = 208.05` руб/мин, что соответствует `fval*100 коп/Int1`/60 с = 208.05*100/80/60 = 4,33 копейки за 1 Мбит, где `Int1 = 80` = $\sum_{i=1}^{8} a_i$ — общая скорость передаваемых данных.

2. Теперь рассмотрим заданный в условии задачи вариант неодинаковой интенсивности передачи данных через коммутационные порты. Очевидно, что через каждый из 64 каналов данные могут передаваться с интенсивностью, равной минимальной величине из двух интенсивностей входного и выходного коммутационных портов, подсоединенных к данному каналу. Произойдет уменьшение общей скорости передаваемых данных с 80 Мбит/с до `Int2 =` 60 Мбит/с. Заметим, что максимальная скорость передачи составляет 66 Мбит/с. Приведенная к полученной интенсивности стоимость передачи по каждому каналу будет определяться командой

```
for i=1:8, for j=1:8
    d2(i, j)=d(i, j)/min (a(i), b(j));
end end
```

Изменив содержание вектора `f = d2(:)`, задающего условие оптимального решения, и снова обратившись к *m*-функции `linprog`, в итоге получим новую расстановку выбранных каналов:

$$d_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} \mathbf{22.49} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{22.46} & 0 \\ 0 & 35.71 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{20.49} \\ 0 & 0 & 0 & 26.85 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 29.55 & 0 & 0 & 0 \\ 0 & 0 & 31.89 & 0 & 0 & 0 & 0 & \mathbf{0} \\ 0 & 0 & 0 & 0 & 0 & \mathbf{20.28} & 0 & 0 \end{bmatrix} \end{array}$$

В матрице d_2 полужирным шрифтом выделены элементы, которые совпадают с минимальными значениями по столбцу в исходной матрице d.

Минимальная стоимость передачи данных по выбранным каналам при неодинаковой интенсивности и перекрестном соединении составила `fval = 209.72` руб/мин, что соответствует `fval*100 коп/Int2/60 с =` = 209.72*100/60/60 = 5,83 копейки за 1 Мбит.

3. Предположим, что существенная часть каналов отключена, так что исходная матрица d стала верхней треугольной, т. е. $d_{ij} = \infty$, $i > j$ (24 канала выключены). Тогда возможности по выбору каналов отсутствуют, поскольку все восемь коммутационных портов должны быть соединены параллельно, согласно своим порядковым номерам. На матрице стоимостей «выбранных» каналов это соответствует главной диагонали. Произойдет дальнейшее уменьшение общей скорости передаваемых данных с 60 Мбит/с до `Int3 =` 56 Мбит/с.

При этом стоимость передачи данных по каналам с одинаковыми номерами портов при неодинаковой их интенсивности существенно возросла и составила по результатам расчетов `fval = 321.08` руб/мин, что соответствует `fval*100 коп/Int3/60 с =` 321.08*100/56/60 = 8,92 копейки за 1 Мбит.

Результаты оптимизации всех трех вариантов оптимального распределения сведены в табл. 1.6.

Таблица 1.6

Итоги оптимизационного анализа системы связи

Вариант 1		Вариант 2		Вариант 3	
№ портов	Скорость	№ портов	Скорость	№ портов	Скорость
(1,1)	10	(1,1)	12	(1,1)	12
(7,2)	10	(3,2)	4	(2,2)	4
(5,3)	10	(7,3)	15	(3,3)	8
(4,4)	10	(5,4)	4	(4,4)	4
(6,5)	10	(6,5)	5	(5,5)	5
(8,6)	10	(8,6)	4	(6,6)	4
(2,7)	10	(2,7)	12	(7,7)	15
(3,8)	10	(4,8)	4	(8,8)	4
208,05 руб	80 Мбит/с	209,72 руб	60 Мбит/с	321,08 руб	56 Мбит/с
4,33 коп/Мбит		5,83 коп/Мбит		8,92 коп/Мбит	

1.3.3. Венгерский алгоритм оптимизации минимальной стоимости

В заключение главы рассмотрим один из популярных алгоритмов «ручного» решения оптимизационных задач о назначениях, использовавшихся в «докомпьютерную эпоху». Это *венгерский алгоритм* — алгоритм оптимизации, решающий задачу о назначениях за полиномиальное время. Он был разработан и опубликован американским математиком Харолдом Куном в 1955 году [8]. Автор дал ему имя «венгерский метод» в связи с тем, что алгоритм в значительной степени основан на более ранних работах венгерских математиков Д. Кенига и Э. Эгервари. Описание этого алгоритма может быть полезным при разработке программ для решения некоторых потоковых задач.

После того как были рассмотрены задачи, сводящиеся к нахождению потока в сети минимальной стоимости, покажем применение венгерского алгоритма для решения этой задачи. Будем считать, что имеется транспортная сеть (см. рис. 1.12), состоящая из двух подмножеств вершин: $x_1, x_2, ..., x_m$ и $y_1, y_2, ..., y_n$, а также истока x_0 и стока y_0. Считаются заданными пропускные способности $a_i = c_{0i}$ и $b_j = c_{j0}$ ветвей, соединяющих соответственно исток x_0 с вершинами x_i и вершины y_j со стоком y_0. Для ветвей (x_i, y_j), соединяющих вершины x_i с вершинами y_j, также заданы пропускные способности c_{ij}. Требуется найти поток φ_{ij}, насыщающий выходные ветви, т. е. такой поток, который удовлетворяет ограничениям

$$\begin{cases} \varphi_{0i} = c_{0i} = a_i; \\ \varphi_{j0} = c_{j0} = b_j; \\ 0 \le \varphi_{ij} \le c_{ij} \end{cases} \tag{1.44}$$

и обращает в максимум линейную форму

$$W(\varphi) = \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} \varphi_{ij} \to \max, \tag{1.45}$$

где d_{ij} — заданные константы, представляющие, например, локальные затраты.

Аналогию с задачей о потоке максимальной стоимости можно получить, если учесть соотношения (1.43). Первые два соотношения в выражении (1.44) означают, что первоначально рассматривается поток,

насыщающий крайние (входные и выходные) ветви, которые показаны на рис. 1.12 входящими в минимальное сечение.

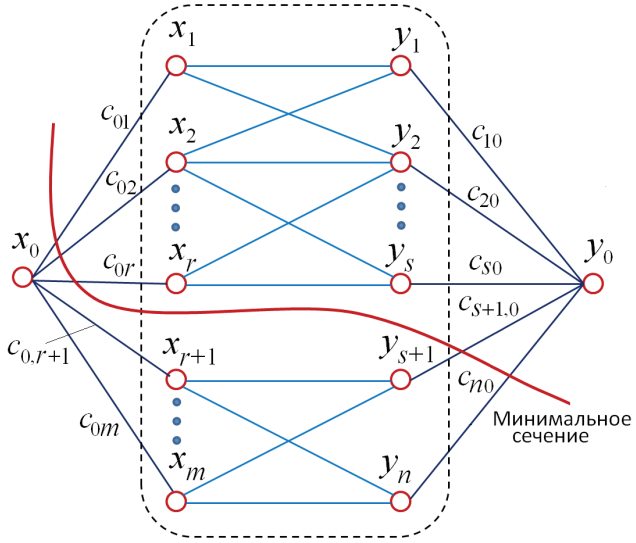


Рис. 1.12. Минимальное сечение на графе задачи о распределении

Выражение (1.45) называется работой потока [1, 8]. При этом величины φ отождествляются с напряжением (иногда говорят потенциалом) некоторого силового поля, а коэффициенты d_{ij} линейной формы — с зарядом источника. Таким образом, требуется найти поток, насыщающий выходные ветви и производящий максимальную работу.

Наряду с основной задачей, задаваемой формулами (1.44) и (1.45), рассмотрим двойственную к ней задачу. Для этого зададим на ветвях сети некоторую положительно-значную функцию, называемую функция-бюджет $\gamma(u)\geqslant 0$, такую, что

$$\gamma(x_0, x_i) + \gamma(x_i, y_j) + \gamma(y_j, y_0) \geq d_{ij}. \qquad (1.46)$$

В двойственной задаче требуется минимизировать линейную форму

$$L(\gamma) = \sum_{i=1}^{m} c_{0i}\gamma(x_0, x_i) + \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\gamma(x_i, y_j) + \sum_{j=1}^{n} c_{j0}\gamma(y_j, y_0) \to \min. \qquad (1.47)$$

Нетрудно проверить правильность составления двойственной задачи, сформировав транспонированную матрицу коэффициентов левой части соотношений в уравнении (1.44). Функция $L(\gamma)$ называется

способностью бюджета γ. Соотношение (1.46) можно пояснить следующим образом. Если трактовать $\gamma(u)$ как количество денег, которое надо уплатить за все транспортные средства перевозки по пути, то соотношение (1.46) означает, что эта величина не должна быть меньше наибольшей полной стоимости работы d_{ij}, которая совершается, когда транспорт курсирует по пути $(x_0 \to x_i \to y_j \to y_0)$.

Итак, как и в теореме о максимальном потоке, если найдется такой поток φ и бюджет γ, что $W(\varphi) = L(\gamma)$, то φ будет искомым решением, которое максимизирует работу.

Введем более удобные обозначения

$$\begin{cases} \gamma(x_0, x_i) = \alpha_i; \\ \gamma(y_j, y_0) = \beta_j; \\ \gamma(x_i, y_j) = \gamma(i, j); \\ \varphi(x_i, y_j) = \varphi(i, j). \end{cases} \tag{1.48}$$

Допустим, что все пропускные способности каналов бесконечны, что означает формальное совпадение позиций x_i и y_j, т. е. затраты в сети отсутствуют. Этот случай встречается в задаче Хичкока (см. пример 1.6). Тогда все бюджеты должны равняться нулю: $c_{ij} = \infty \to \gamma_{ij} = 0$. Теперь двойственная задача заключается в том, чтобы найти такие числа $\alpha_i \geq 0$ и $\beta_j \geq 0$, для которых $\alpha_i + \beta_j \geq d_{ij}$. Формально в неравенствах (1.46) и (1.47) исключены средние слагаемые, что существенно упрощает задачу оптимизации. Для последующего построения венгерского алгоритма рассмотрим вспомогательную задачу: каждому бюджету γ отнесем вспомогательную транспортную сеть $\overline{G}_\gamma$, которая получается из исходной сети G_γ удалением тех внутренних ветвей (x_i, y_j), для которых $\alpha_i + \beta_j \geq d_{ij}$.

Исходим из бюджета $\gamma_{ij} = 0$:

$$\begin{cases} \alpha_i = \max\limits_j d_{ij}; \\ \beta_j = \max\limits_i \left[0, \left(d_{ij} - \alpha_i\right)\right] = \max\limits_i \left[0, \left(d_{ij} - \max\limits_j d_{ij}\right)\right]. \end{cases} \tag{1.49}$$

Следствием соотношений (1.49) является равенство $\alpha_i + \beta_j = 0$, поэтому конечные ветви (x_{0i}, y_{j0}) могут войти в сеть $\overline{G}_\gamma$. Далее ищется максимальный поток в этом графе. Если он насыщает выходные ветви, то задача решена. Если не насыщает, то по формулам

$$\alpha'_i = \begin{cases} \alpha_i, & i \le r; \\ \alpha_{i-1}, & i > r; \end{cases} \qquad \beta'_j = \begin{cases} \beta_j, & j \le s; \\ \beta_{j+1}, & j > s \end{cases} \tag{1.50}$$

определяется новый бюджет γ' и процедура повторяется. Для пояснения формулы (1.50) следует обратиться к рис. 1.12. Сечение пересекает начальные ветви при $i \leqslant r$ и конечные ветви при $j > s$, внутренние ветви (x_i, y_j) при $i \leqslant r$ и $j > s$ в сечение не входят. Другие внутренние ветви входят в сеть $\overline{G}_\gamma$ с номерами $(i \le r, j > s)$, $(i \le r, j < s)$, $(i > r, j > s)$, если только $\alpha_i + \beta_j = d_{ij}$.

Аналогичная процедура строится для конечных значений пропускных способностей каналов (внутренних ветвей) при $c_{ij} \neq \infty$. Можно принять

$$\gamma_{ij} = \max_i \left\{0, \left(d_{ij} - \alpha_i - \beta_j\right)\right\}. \tag{1.51}$$

Тогда при $\alpha_i = \max\limits_j d_{ij}$, как и в случае бесконечных способностей, величина

$$\beta_j = \max_i \left\{0, \left(d_{ij} - \alpha_i\right)\right\}. \tag{1.52}$$

При этом все бюджетные величины γ_{ij} неотрицательны. Тогда возможны три случая согласно соотношению (1.46):

$$\begin{cases} 1)\ \gamma_{ij} = 0,\ d_{ij} = \alpha_i + \beta_j, & (x_i, y_j) \in S; \\ 2)\ \gamma_{ij} = 0,\ d_{ij} \le \alpha_i + \beta_j, & (x_i, y_j) \in Q; \\ 3)\ \gamma_{ij} > 0,\ d_{ij} = \alpha_i + \beta_j + \gamma_{ij}, & (x_i, y_j) \in R. \end{cases} \tag{1.53}$$

Вспомогательная транспортная сеть $\overline{G}_\gamma$ имеет те же ветви, что и сеть G, но пропускные способности будут отличаться:

$$\begin{cases} \bar{a}_i = a_i - \sum\limits_{j/(i,j)\in R} c_{ij},\ i = \overline{1,m}; \\ \bar{b}_j = b_j - \sum\limits_{i/(i,j)\in R} c_{ij},\ j = \overline{1,n}; \\ \bar{c}_{ij} = \begin{cases} c_{ij}, & (i,j) \in S; \\ 0, & (i,j) \in Q \cup R. \end{cases} \end{cases} \tag{1.54}$$

Суммирование в первой формуле (1.54) проводится при фиксированном индексе i по всем j таким, что ветвь (i, j) или (x_i, y_j) принадлежит подмножеству R, аналогично во второй формуле (1.54) — при фиксированном j по тем i, у которых ветвь (x_i, y_j) принадлежит подмно-

жеству R. В третьей формуле системы (1.54) $\bar{c}_{ij} = 0$ тогда, когда ветвь (x_i, y_j) принадлежит хотя бы одному из подмножеств Q и R или обоим вместе. При этом можно считать, что все $\alpha_i > 0$.

Для определения нового бюджета (при отсутствии насыщения граничных ветвей) следует, как при обращении к формуле (1.50), найти новые числа

$$\left(\gamma_{ij}\right)' = \begin{cases} \gamma_{i-1,j}, & i \le r, j > s, (ij) \in R; \\ \gamma_{i,j+1}, & i > r, j \le s, (ij) \in R \cup S; \\ \gamma_{ij}, & \text{в остальных случаях}; \end{cases}$$

$$\alpha_i' = \begin{cases} \alpha_i, & i \le r; \\ \alpha_{i-1}, & i > r; \end{cases} \quad \beta_j' = \begin{cases} \beta_j, & j \le s; \\ \beta_{j+1}, & j > s, \end{cases} \tag{1.55}$$

которые должны удовлетворять ограничению $\alpha_i' + \beta_j' + (\gamma_{ij})' - d_{ij} \ge 0$.

Представим суть венгерского алгоритма в формальном изложении. Пусть заданы два конечных множества K и L и матрица a [K, L] (обычно считается, что $|K| = |L|$). Цель — найти взаимно однозначное соответствие $f : K \to L$, которое делает минимальной сумму от $a[k, f(k)]$ по всем k. Алгоритм не кажется особо сложным (см. рис. 1.13), хотя не просто понять, почему он работает. Алгоритм состоит из трех основных шагов [1, 8].

Шаг 1. «Опускается» до нуля каждая строка и каждый столбец. Это означает, что вводится по крайней мере один ноль в каждой строке и каждом столбце матрицы $a[k, l]$.

Шаг 2. Строится максимальное паросочетание на всех нулях, полученных на предыдущем шаге.

Шаг 3. Если максимальное паросочетание окажется полным, то задача уже решена. В противном случае вместе с паросочетанием строится минимальное контролирующее множество (K', L'), которое помогает решить задачу. Согласно определению минимального контролирующего множества для всех нулевых элементов $a[k, l]$ либо k принадлежит K', либо l принадлежит L'. Поэтому все элементы подматрицы a [$K \backslash K'$, $L \backslash L'$] больше нуля. Находится минимальный элемент этой подматрицы и прибавляется к a [$K \backslash K'$, L] и вычитается из a [K, $L \backslash L'$]. После этого выполняется переход на шаг 2. Алгоритм выполняется до тех пор пока не будет построено полное паросочетание.

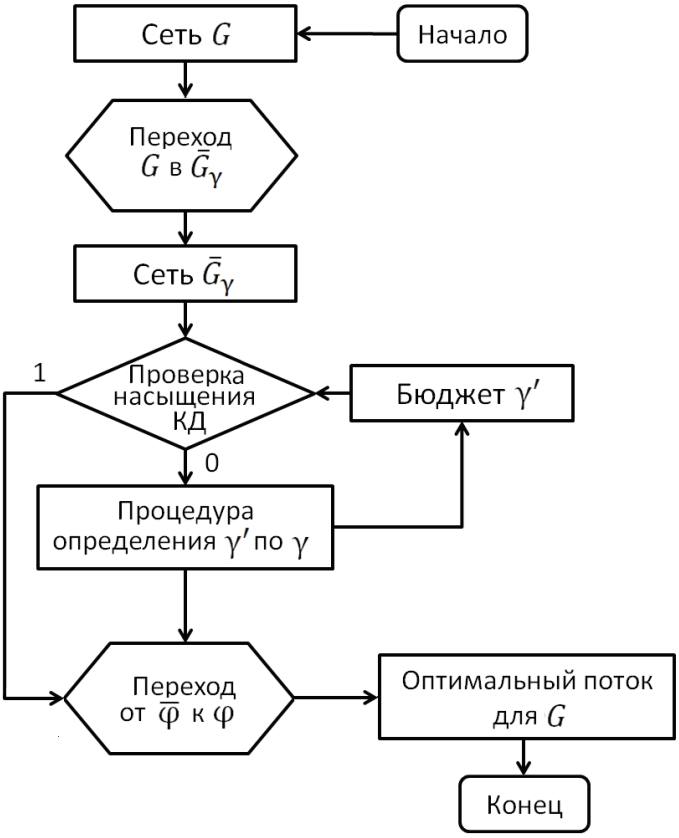


Рис. 1.13. Блок-схема венгерского алгоритма

Пример 1.10. Поиск максимального потока в сети с помощью венгерского алгоритма.

Рассмотрим сеть, изображенную на рис. 1.14, *а*. Зададим коэффициенты линейной формы d_{ij} в виде табл. 1.7.

Таблица 1.7

Затраты в сети и их сравнение с пропускными способностями ветвей

i	d_{ij}		
	$j=1$	$j=2$	$j=3$
1	2	3	1
2	4	2	3
3	1	5	1
4	3	1	2

i	D_{ij}		
	$j=1$	$j=2$	$j=3$
1	<0	0	<0
2	0	<0	<0
3	<0	0	<0
4	0	<0	<0

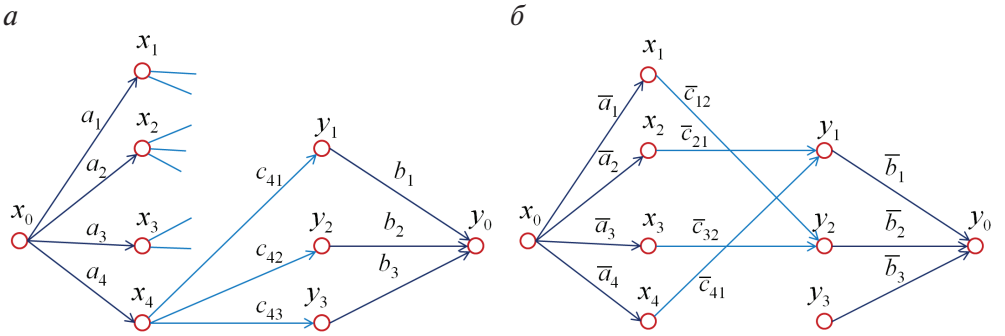


Рис. 1.14. Вспомогательный граф сети $\overline{G}_\gamma$ в венгерском алгоритме

Используем соотношения (1.51) для определения величин α_i, которые равны максимальным значениям d_{ij} в каждой i-й строке. Все величины β_j и γ_{ij} согласно уравнению (1.52) будут равны нулю, так как при всех значениях (ij) разности $d_{ij} - \alpha_i \leqslant 0$. Итак, первичный набор элементов α_i, β_i, γ_{ij} равен:

$$\alpha_i = [3, 4, 5, 3],\ i = \overline{1,4};\ \beta_i = \gamma_{ij} = 0.$$

Далее с помощью формул (1.53) определим множества S, Q, R. Для этого составим таблицу значений разности $D_{ij} = d_{ij} - \alpha_i - \beta_j$, которые отображены в правой части табл. 1.7. В этой таблице находится четыре нулевых элемента, которые определяют элементы множества S. Все остальные элементы меньше нуля, поэтому они входят в множество Q. Элементов $D_{ij} > 0$ в таблице нет, так что множество R пустое.

$$S = \{(x_1, y_2), (x_2, y_1), (x_3, y_2), (x_4, y_1)\};$$
$$Q = \{\text{оставшиеся ветви}\};$$
$$R = \varnothing \text{ (пустое множество).}$$

Пропускные способности ветвей сети $\overline{G}_\gamma$ находятся в соответствии с третьей формулой системы (1.54):

$$\overline{a}_{0i} = \overline{c}_{0i} = a_i,\quad i = \overline{1,4};\quad \overline{b}_{j0} = \overline{c}_{j0} = b_j,\quad j = \overline{1,3};$$
$$\overline{c}_{14} = \overline{c}_{13} = \overline{c}_{22} = \overline{c}_{23} = \overline{c}_{31} = \overline{c}_{33} = \overline{c}_{42} = \overline{c}_{43} = 0;$$
$$\overline{c}_{12} = c_{12};\quad \overline{c}_{21} = c_{21};\quad \overline{c}_{32} = c_{32};\quad \overline{c}_{41} = c_{41}.$$

Сеть $\overline{G}_\gamma$, соответствующая этим значениям, изображена на рис. 1.14, *б*.

В этой сети с помощью алгоритма Форда-Фалкерсона необходимо определить максимальный поток. Если этот поток насыщает выходные дуги, то задача решена. Если насыщения выходных дуг в сети $\overline{G}_\gamma$ не происходит, то по формулам системы (1.55) изменяют значения двойственных переменных и повторяют процедуру отыскания максимального потока, и т. д.

1.3.4. Динамические сетевые модели

Исследования в теории потоков были прежде всего мотивированы военными нуждами благодаря связи между максимальными потоками и минимальными сечениями. Решение задачи о минимальном сечении позволяло построить эффективный план бомбардировок системы транспортного сообщения противника. Помимо этого с практической точки зрения была крайне важна задача об эвакуации на случай бомбардировок или чрезвычайных происшествий.

Мирным применением теории потоков являются всевозможные задачи, связанные с транспортировкой грузов. В более простой модели ответ на этот вопрос дает задача о назначениях — обобщение задачи о максимальном паросочетании. Более сложные модели опираются на теорию динамических потоков, и здесь есть еще много задач для дальнейших исследований.

Выше были рассмотрены модели сетей с неизменными во времени параметрами и время переноса какого-либо объекта (топлива, данных, заявок и т. п.) в таких сетях не задавалось. Форд и Фалкерсон понимали важность времени в транспортировке и учли его в их модифицированной модели сети. Они обобщили стандартное определение сети путем добавления в него транзитных времен между вершинами, получив тем самым динамическую сеть. Каждая ветвь (x_i, y_j) в динамической сети моделирует трубопровод из вершины x_i в вершину y_j. Пропускная способность ветви (x_i, y_j) соответствует площади сечения труб, тем самым ограничивая количество потока, которое может быть передано за единицу времени. Транзитные времена соответствуют длине трубопровода, они определяют, сколько времени требуется потоку на прохождение из вершины x_i в вершину y_j по ветви (x_i, y_j). Динамический поток перемещается с течением времени в динамической сети.

Он является расширением традиционных потоков: отображением пар (ветвь, время) в величине потока, а не отображением только ветвей, как это было ранее.

Другой вариант задачи о максимальном динамическом потоке — это задача о наибыстрейшем потоке, в которой задана величина потока v и требуется найти допустимый динамический поток, который доставит v единиц потока из истока в сток за наименьшее возможное время. Задача о наибыстрейшем потоке может быть сведена к задаче о максимальном динамическом потоке с помощью двоичного поиска. Баркард (Burkard) и др. изучали эту задачу и описали более эффективную технику ее решения [9].

Задача эвакуации является обобщением задачи о наибыстрейшем потоке на случай нескольких источников. Для каждого из них задан вектор снабжения и требуется найти допустимый динамический поток, который доставляет требуемое количество потока из каждого источника в сток за наименьшее возможное время. В традиционной сети без значений времени транзита задача эвакуации тривиально сводится к задаче о максимальном потоке с одним источником (путем добавления «супериточника»). Однако, в динамическом случае задача является гораздо более сложной, чем задача о наибыстрейшем потоке. В то же время она полезнее: задача эвакуации была изначально сформулирована в практической модели эвакуации людей из зданий.

Структура потоковых задач ведет к гораздо более эффективным решениям (как с теоретической, так и с практической точек зрения), чем решение линейных программ. И этот подход получил наибольшее внимание со стороны исследователей с тех пор, как Форд и Фалкерсон выбрали его в своем фундаментальном труде по потокам в сетях [3]. В течение последующих четырех десятилетий одной из основных задач для сообщества исследователей в области компьютерных наук и исследования операций стало повышение эффективности алгоритмов для потоков в сетях. Решенной эту задачу можно считать с момента появления алгоритма Рао-Гольдберга в 1997 году [10]. С тех пор основные усилия исследователей направлены на изучение динамических потоков — обобщения обычных («статических») потоков, учитывающих время.

Вопросы и задания к главе 1

1. Дайте определение транспортной сети. Какие обязательные условия в ней должны быть выполнены?
2. Что означает отсутствие петель в транспортной сети и каким образом следует формализовать это условие?
3. Из каких графических объектов состоит граф транспортной сети?
4. Дайте определение пропускной способности и потока ветви в графе транспортной сети. В чем состоит их сходство и различие?
5. Всегда ли поток, исходящий из истока транспортной сети, должен совпадать с потоком, входящим в ее сток?
6. Всегда ли суммарная пропускная способность входных ветвей должна быть равна суммарной способности выходных ветвей транспортной сети?
7. Дайте определение сечения в графе транспортной сети. Какие условия накладываются на выделение в сети канонического сечения?
8. Сформулируйте и поясните роль теоремы Форда-Фалкерсона в теории графов транспортных сетей.
9. Что означает потенциальная разбалансированность транспортной сети и каким образом она может быть устранена?
10. Поясните соответствие структуры и параметров транспортной сети и коэффициентов уравнений системы прямой задачи линейного программирования.
11. Какую часть информации о транспортной сети следует связать с системой равенств прямой задачи линейного программирования?
12. Составьте систему равенств прямой задачи линейного программирования для транспортной сети, изображенной на рис. 1.15.
13. Составьте систему неравенств прямой задачи линейного программирования для транспортной сети, изображенной на рис. 1.15.
14. Запишите вектор линейной формы прямой задачи линейного программирования для транспортной сети, изображенной на рис. 1.15.
15. С помощью *m*-функции `linprog` определите максимальный поток в сети на рис. 1.15 и выделите на графе сети критическое сечение.
16. По правилам, указанным в п. 1.2.5, подготовьте текстовый файл с описанием транспортной сети, изображенной на рис. 1.15, и с помощью скрипт-файла TransNet.m найдите максимальный поток.

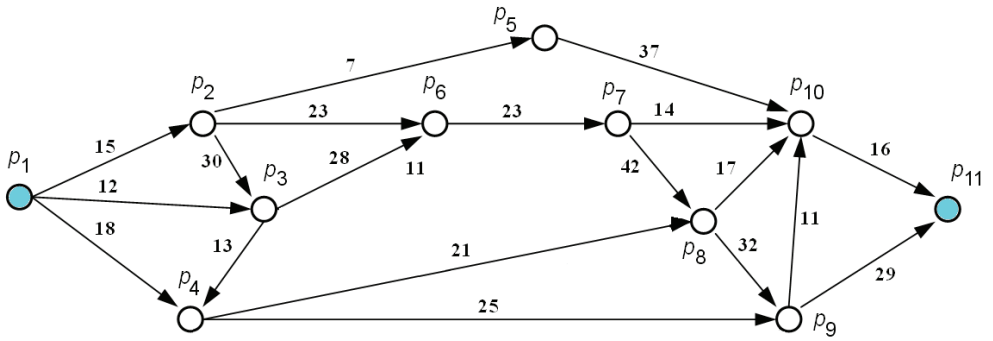


Рис. 1.15. Потенциально сбалансированная транспортная сеть

17. Сравните величины максимальных потоков, полученные в п. п. 15 и 16 и в случае несовпадения найдите и исправьте ошибки при задании данных.
18. Три шахты и три теплоцентрали связаны транспортной системой, имеющей следующую матрицу стоимости перевозок (в тыс. руб. за тонну)

$$C = \begin{bmatrix} 5 & 8 & 10 \\ 3 & 11 & 6 \\ 7 & 6 & 9 \end{bmatrix} \begin{matrix} \text{Шахта 1} \\ \text{Шахта 2}. \\ \text{Шахта 3} \end{matrix}$$

Столбцы матрицы C относятся к теплоцентралям. Шахты выдают «на гора» по 160 тонн в сутки, а теплоцентрали потребляют в сутки 120, 150 и 210 тонн угля соответственно. Составьте оптимальный по минимуму общих транспортных затрат план доставки топлива.

19. Измените условия задачи в п. 18 на обратные: теплоцентрали потребляют одинаковое число тонн угля, а шахты отгружают в сутки 120, 150 и 210 тонн угля соответственно. Каким образом изменится оптимальный план перевозок?

2. Модели на основе сетей Петри

Современные вычислительные системы невозможно представить без устройств, обеспечивающих обмен информацией между отдельными компьютерами, т. е. без локальных или корпоративных сетей передачи данных. Более того, в современных условиях даже отдельный компьютер без подключения к глобальной сети — Интернету — часто становится неэффективной и почти ненужной вещью. Поэтому ознакомление с процессами функционирования сетей передачи данных в форме создания и последующего анализа их моделей необходимо специалисту по информационным технологиям и желательно для каждого грамотного компьютерного пользователя.

Отличительной особенностью процессов в любой сети является одновременное прохождение информационных и служебных потоков данных по ее различным фрагментам. Для описания логики работы сетевой модели могут быть использованы различные средства: либо русский язык (устный или письменный), либо традиционные схемы алгоритмов, либо какие-то другие инструментальные средства. Первые два варианта являются, как правило, наиболее знакомыми и часто используемыми. Однако описание в виде блок-схемы алгоритма модели даже простой сети может оказаться затруднительным прежде всего потому, что такие схемы слабо приспособлены для описания параллельных процессов.

Именно поэтому представляется необходимым познакомить студента с одним из наиболее элегантных и весьма распространенных средств описания параллельных процессов — сетями Петри (англ. Petri Nets, предложены К. Петри [11, 12, 13]). Им посвящено достаточно большое число публикаций, поэтому ограничимся изложением только тех основных сведений, которые необходимы с точки зрения реализации технологии имитационного моделирования параллельных процессов.

2.1. Примитивные сети Петри

Одно из основных достоинств аппарата сетей Петри (сокращенно *N*-сети) заключается в том, что они могут быть представлены как в графической форме (это обеспечивает наглядность), так и в аналитической (это позволяет автоматизировать процесс их анализа). При графической интерпретации сеть Петри представляет собой граф особого вида, состоящий из вершин двух типов — позиций P и переходов T, соединенных ориентированными ветвями (множества I и O, характеризующие начала и концы ветвей), причем каждая ветвь может связывать лишь разнотипные вершины (позицию с переходом или переход с позицией). Вершины-позиции P обозначаются точками или кружками, вершины-переходы T — черточками или узкими прямоугольниками (рис. 2.1).

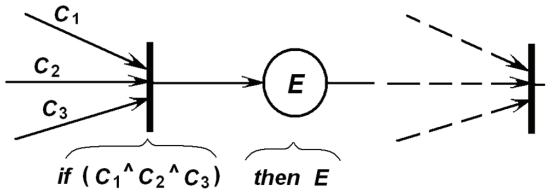
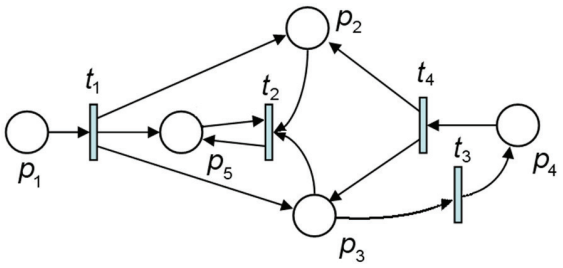


Рис. 2.1. К определению *N*-сети

С содержательной точки зрения переходы соответствуют событиям, присущим исследуемой системе, а позиции — условиям их возникновения. Таким образом, совокупность переходов, позиций и ветвей графа позволяет наглядно описать причинно-следственные связи, присущие системе, но в статике (рис. 2.2, *а*).

а



б

$$A = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & -1 & 1 \\ 0 & 0 & 1 & -1 \\ 1 & \pm 1 & 0 & 0 \end{bmatrix}$$

Рис. 2.2. Классический пример графа *N*-сети — *а* и его матрица инциденций — *б*

Формально простая сеть Петри (*N*-схема) задается четверкой вида [11, 12]

$$N = < P, T, I, O >, \tag{2.1}$$

где P — конечное непустое множество символов-позиций; T — конечное непустое множество символов-переходов; I — входная функция (прямая функция инцидентности); O — выходная функция (обратная функция инцидентности). Таким образом, входная функция I отображает переход t_j во множество входных позиций $p_i \in I(t_j)$, а выходная функция O отображает переход t_j во множество выходных позиций $p_i \in O(t_j)$. Для каждого перехода $t_j \in T$, $j = 1,2,\ldots,m$, можно определить множество входных позиций перехода $I(t_j)$ и выходных позиций перехода $O(t_j)$ как

$$\begin{aligned} &I(t_j) = \{p_i \in P \mid I(p_i, t_j) = 1\}, \\ &O(t_j) = \{p_i \in P \mid O(t_j, p_i) = 1\}, \quad i = \overline{1,n}, \ j = \overline{1,m}, \ n = |P|, \ m = |T|. \end{aligned} \tag{2.2}$$

Входная и выходная функции сети Петри (I и O) позволяют описать любую сеть с помощью двух матриц размером $n \times m$ (матриц входных и выходных позиций), имеющих следующую структуру (состоят только из положительных единиц и нулей):

$$I = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ \cdots \\ p_n \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & \cdots & t_m \end{array} \\ \begin{bmatrix} 1 & 0 & \cdots & 1 \\ 0 & 1 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 0 & \cdots & 1 \end{bmatrix} \end{array}, \qquad O = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ \cdots \\ p_n \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & \cdots & t_m \end{array} \\ \begin{bmatrix} 0 & 0 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ 0 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 0 & \cdots & 0 \end{bmatrix} \end{array}.$$

Если ориентацию ветвей сделать одинаковой, например, положительным направлением (t_j, p_i) считать выход из перехода t_j и вход в позицию p_i, то соотношения (2.2) можно представить как алгебраическую запись матрицы инциденций (см. рис. 2.2, *б*), т. е. объединить обе матрицы позиций в одну:

$$A_{ij} = \begin{cases} -I(p_i, t_j) \cup O(t_j, p_i), & I(p_i, t_j) \times O(t_j, p_i) = 0; \\ \pm 1, & I(p_i, t_j) \times O(t_j, p_i) \neq 0. \end{cases} \tag{2.3}$$

Первая строка в выражении (2.3) соответствует одной ветви, связывающей i-ю позицию и j-й переход, вторая строка — двум встречно на-

правленным ветвям (петля из двух ветвей, как, например, на рис. 2.2, а между позицией p_5 и переходом t_2).

Аналогично соотношениям (2.2) и (2.3) для каждой позиции $p_i \in P$ вводятся определения множества входных переходов позиции $I(p_i)$ и множества выходных переходов позиции $O(p_i)$:

$$I(p_i) = \{t_j \in T \mid I(t_j, p_i) = 1\},$$
$$O(p_i) = \{t_j \in T \mid O(p_i, t_j) = 1\}, \tag{2.4}$$

а матрица инциденций A_{ji}, составленная по выражению (2.4), есть транспонированная матрица A_{ij}. Обе записи выражений (2.2) и (2.4) эквивалентны, в дальнейшем будет использоваться первое определение, в котором строки матрицы A_{ij} означают позиции $p_i \in P$, а столбцы — переходы $t_j \in T$ (см. рис. 2.1, *б*). В матрице А не должно быть полностью нулевых строк или столбцов, означающих отсутствие каких-либо ветвей у соответствующих позиций и переходов.

Графически сеть Петри изображается в виде двудольного ориентированного мультиграфа [11], представляющего собой совокупность позиций и переходов (см. рис. 2.2). Как видно из этого рисунка, граф *N*-схемы имеет два типа вершин: позиции и переходы, изображаемые *P* и *T* соответственно. Ориентированные ветви $a_{ji} = (t_j, p_i)$ соединяют позиции и переходы, причем каждая ветвь направлена от элемента одного множества (позиции или перехода) к элементу другого множества (переходу или позиции). Граф *N*-схемы является мультиграфом, так как он допускает существование кратных ветвей от одной вершины к другой.

Пример 2.1. Описание *N*-сетей с помощью множеств

Составим определенные выше множества для *N*-схемы (см. рис. 2.2), часто используемой в качестве примера сети Петри:

- множество позиций $P = \{p_1, p_2, p_3, p_4, p_5\}$;
- множество переходов $T = \{t_1, t_2, t_3, t_4\}$;
- входное множество для переходов $I(t_1) = \{p_1\}$,
  $I(t_2) = \{p_2, p_3, p_5\}$,
  $I(t_3) = \{p_3\}$,
  $I(t_4) = \{p_4\}$;

• выходное множество для переходов $O(t_1)=\{p_2,p_3,p_5\}$,

$O(t_2)=\{p_5\}$,

$O(t_3)=\{p_4\}$,

$O(t_4)=\{p_2,p_3\}$.

Альтернативное описание связей между позициями и переходами можно выполнить, используя входные и выходные множества для позиций:

• входное множество для позиций $I(p_1)=\{\ \}$,

$I(p_2)=\{t_1,t_4\}$,

$I(p_3)=\{t_1,t_4\}$,

$I(p_4)=\{t_3\}$,

$I(p_5)=\{t_1,t_2\}$;

• выходное множество для позиций $O(p_1)=\{t_2\}$,

$O(p_2)=\{t_2\}$,

$O(p_3)=\{t_2,t_3\}$,

$O(p_4)=\{t_4\}$,

$O(p_5)=\{t_2\}$.

Описание этой сети Петри с помощью множеств (функций) для переходов $I(t_j)$ и $O(t_j)$ оказалось экономнее, чем с помощью множеств для позиций $I(p_i)$ и $O(p_i)$. Кроме того, одно из входных множеств позиций — множество $I(p_1)$ — оказалось пустым. Выбор того или иного варианта зависит от вида конкретной сети Петри и цели моделирования.

2.1.1. Моделирование динамики в сети Петри

Приведенные выше представления *N*-схемы в виде множеств, графа и матриц могут использоваться только для отображения статических состояний моделируемой системы (взаимосвязи событий и условий), описывая ее, по-существу, в форме структурной схемы. Для отражения в модели динамики функционирования моделируемой системы вводят еще один вид объектов сети Петри — так называемые фишки

или *метки* m_i позиций $p_i \in P$, $i = \overline{1,n}$. Процесс присвоения неких абстрактных объектов — меток — позициям *N*-схемы называется *маркировкой* и обозначается символом *M*, а ее результат — положение меток в позициях сети — называется *разметкой* сети. Количество меток, соответствующее каждой позиции, может меняться. При графическом задании *N*-схемы разметка отображается помещением внутри вершин-позиций соответствующего числа точек (когда количество точек велико, ставят цифры).

Функция маркировки $M(p_i)$ определяет число меток m_i, относящихся к каждой *i*-й позиции сети Петри: $m_i \in M \to \{0, 1, 2, \ldots\}$. Размеченная *N*-схема может быть описана в виде пятерки объектов [11, 12]

$$N =< P, T, I, O, M > \tag{2.5}$$

и является совокупностью сети Петри и маркировки *M*.

Функционирование *N*-схемы отображается путем перехода от разметки к разметке. Начальная разметка обозначается как M_0. Смена разметок происходит в результате срабатывания одного из переходов $t_j \in T$ сети. Необходимым условием срабатывания перехода t_j является наличие меток в каждой его входной позиции: $\forall\ p_i \in I(t_j) : \{M(p_i) \geq 1\}$, где $M(p_i)$ — разметка позиции p_i. Переход t_j, для которого выполняется указанное условие, определяется как находящийся в состоянии готовности к срабатыванию или как активный переход.

Срабатывание перехода t_j на некотором *k*-м шаге изменяет разметку сети $M_k(p) = \{M_k(p_i)\}$ на разметку $M_{k+1}(p)$ по следующему правилу:

$$M_{k+1}(p) = M_k(p) - I(t_j) + O(t_j), \tag{2.6}$$

т. е. переход t_j изымает по одной метке из каждой своей входной позиции и добавляет по одной метке в каждую из выходных позиций. Для изображения смены разметки $M_k(p)$ на $M_{k+1}(p)$ при срабатывании перехода t_j применяют обозначение $M_k \vert^{t_j} M_{k+1}$. Таким образом, сеть Петри изменяет первоначальную разметку M_0 путем последовательного запуска переходов под управлением количества меток и их распределения в сети. Переход запускается удалением меток из его входных позиций и образованием новых меток, помещаемых в выходные позиции. Переход может запускаться только тогда, когда он разрешен. Для исключения конфликтных ситуаций, когда в некоторый дискретный момент времени несколько переходов находятся в активном состоянии, долж-

но быть задано *правило выбора* срабатывающего перехода. Таким правилом может быть, например, один из следующих вариантов:

а) выбор перехода с наименьшим (или наибольшим) номером;
б) выбор перехода с наименьшим (или наибольшим) количеством меток;
в) случайный выбор.

Функционирование сети Петри продолжается до тех пор, пока существует хотя бы один активный переход.

Пример 2.2. Динамическая модель сети Петри [11, 12].

Зададим начальную маркировку $M_0 = \{1, 2, 0, 0, 1\}$ для представленной на рис. 2.2 сети Петри и расположим заданные четыре метки внутри кружков, обозначающих соответствующие позиции (см. рис. 2.3, *а*). Выберем правило а) — первым из активных выбирается переход с наибольшим номером.

Из четырех переходов активным является только переход t_1, а остальные переходы t_2, t_3 и t_4 — запрещенные. В результате выполнения этого перехода получим новую размеченную сеть Петри (см. рис. 2.3, *б*) с разметкой $M_1 = \{0, 3, 1, 0, 2\}$. Теперь активны переходы t_2 и t_3, и запущен может быть только один из них. Согласно выбранному правилу запускается переход с наибольшим номером, т. е. переход t_3. В результате его запуска получится новая разметка сети $M_2 = \{0, 3, 0, 1, 2\}$, изображенная на рис. 2.3, *в*. Активным переходом на следующем шаге будет только один — переход t_4. После его срабатывания (см. рис. 2.3, *г*) разметка сети будет такой: $M_3 = \{1, 4, 1, 0, 2\}$. При такой разметке активным является также только один переход t_2, после срабатывания которого разметка $M_4 = \{0, 3, 0, 0, 2\}$, изображенная на рис. 2.3, *д*, остается неизменной, так как ни один из четырех переходов не стал активным. Для выбранного правила эта разметка стала финальной.

Выберем другой вариант разрешения конфликта при нескольких активных переходах, а именно — первым из активных выбирается переход с наименьшим номером. В этом случае первый шаг модификации сети Петри будет совпадать с первым шагом, выполненным при прежнем правиле (сравни рис. 2.4, *б* с рис. 2.3, *б*). Однако правило выбора на втором шаге приведет к срабатыванию активного перехода t_2 (номер перехода t_2 меньше, чем номер перехода t_3). В результате получим сеть, показанную на рис. 2.4, *в*. Теперь ни один переход не может быть запущен и динамическая маркировка сети прекращается.

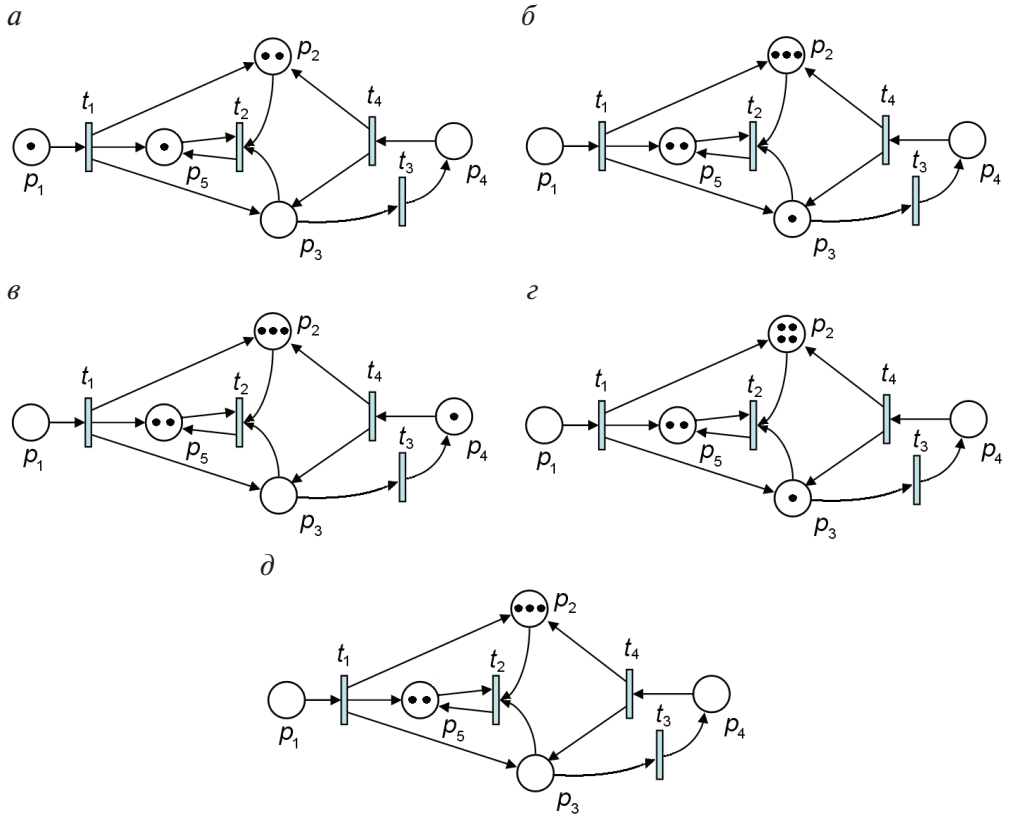


Рис. 2.3. Динамика сети Петри при правиле выбора перехода с наибольшим номером

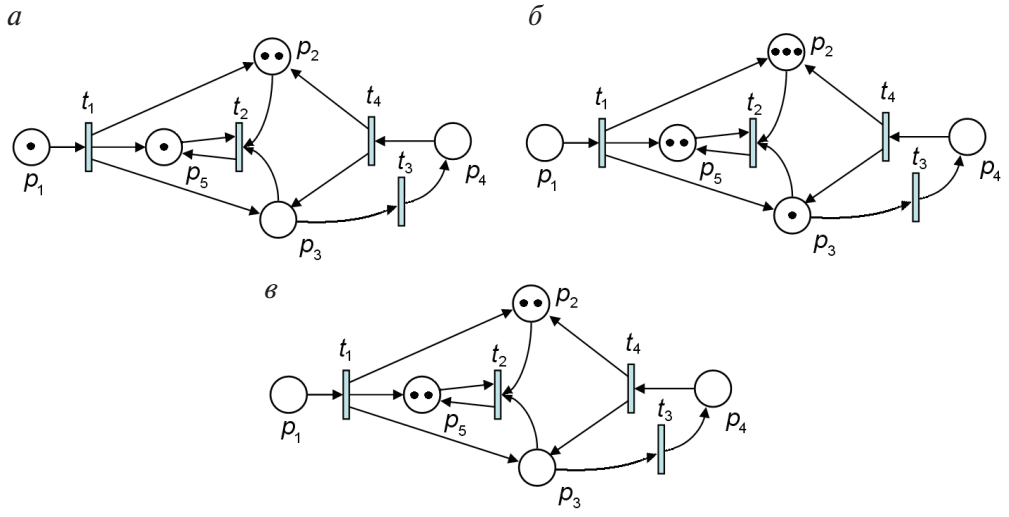


Рис. 2.4. Динамика сети Петри при правиле выбора перехода с наименьшим номером

2.1.2. Методология моделирования систем с помощью сетей Петри

Важной особенностью моделей процесса функционирования систем с использованием типовых *N*-схем является простота построения иерархических конструкций модели. С одной стороны, каждая *N*-схема может рассматриваться как макропереход или макропозиция модели более высокого уровня. С другой стороны, переход или позиция *N*-схемы могут детализироваться в форме отдельной подсети для более углубленного исследования процессов в моделируемой системе (рис. 2.5). Отсюда вытекает возможность эффективного использования *N*-схем для моделирования параллельных и конкурирующих процессов в различных системах. Речь идет о так называемых вложенных сетях Петри (*Nested Petri Nets — NPN*) [12, 14]. Появление указанной разновидности сетей Петри связано с желанием исследователей иметь инструмент для адекватного и удобного представления систем со сложной иерархической и мультиагентной структурой.

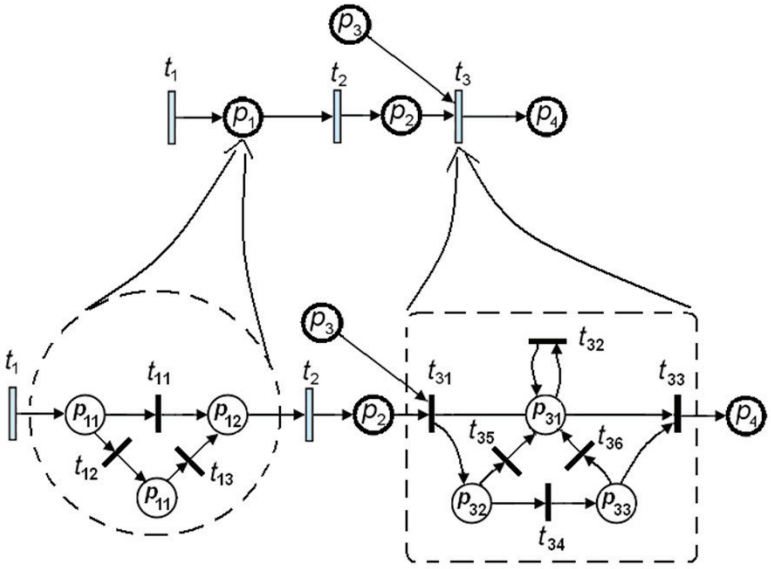


Рис. 2.5. Иерархические расширения сети Петри

Вложенные сети Петри представляют собой расширение стандартного формализма сетей Петри, в котором метки, представляющие локальные ресурсы в позициях системной сети, сами могут быть слож-

ными объектами с сетевой структурой и моделироваться сетями Петри нижнего уровня — их мы также будем называть *сателлитными* сетями *ENi*.

Структурно такая сеть состоит из системной *N*-сети и набора сетей-меток (сателлитов). При этом между некоторыми переходами системной сети и переходами сетей-меток может быть установлена связь, разрешающая только их совместное срабатывание. Такие переходы называются *помеченными*.

Функционирование *NPN*-сетей в значительной мере совпадает с функционированием ранее рассмотренных сетей Петри. Отличие составляют механизмы синхронизации работы сетей Петри различного уровня. В связи с этим в *NPN*-сетях различают следующие четыре вида шагов срабатывания:

- системно-автономный шаг, который соответствует срабатыванию непомеченного перехода в системной *SN*-сети;
- сателлитно-автономный шаг, который соответствует срабатыванию непомеченного перехода в сателлитной *ENi*-сети;
- шаг горизонтальной синхронизации, при котором одновременно срабатывают переходы в сателлитных *ENi*-сетях, помеченные одинаковыми метками;
- шаг вертикальной синхронизации, при котором одновременно срабатывают переходы в системной *SN*-сети и сателлитных *ENi*-сетях, имеющие одинаковые метки.

Разумеется, при этом предполагается, что во всех сетях все участвующие в работе переходы являются активными, т. е. в их входных позициях имеются необходимые для срабатывания ресурсы (выполняются условия возникновения событий).

Итак, достоинства применения сетей Петри для изучения процессов в сложных системах заключаются в следующем:

- позволяют моделировать параллельные процессы всех возможных типов с учетом возможных конфликтов между ними (универсальность);
- обладают наглядностью и обеспечивают возможность автоматизированного анализа (эффективность);
- позволяют переходить от одного уровня детализации описания системы к другому за счет раскрытия или закрытия позиций и переходов (иерархичность).

Вместе с тем сети Петри имеют ряд недостатков, ограничивающих их возможности. Основной недостаток — время срабатывания перехода считается равным нулю, что не позволяет исследовать с помощью простых сетей Петри временные характеристики моделируемых систем. Хотя типовые *N*-схемы на основе обычных размеченных сетей Петри пригодны для описания в моделируемой системе событий произвольной длительности, но в этом случае модель отражает только порядок наступления событий в исследуемой системе. Для отражения временных параметров процесса функционирования моделируемой системы на базе *N*-схем следует использовать различные расширения аппарата сетей Петри: временные сети, *E*-сети, сети Мерлина и т. п. [11].

N-сети с мгновенным выполнением переходов называются *примитивными*. В них возникновение двух событий одновременно невозможно. Непримитивными называются такие события, длительность которых отлична от нуля. Любое непримитивное событие может быть представлено в виде двух примитивных событий, определяющих начало и конец непримитивного события и состояния, когда это непримитивное событие происходит.

Одним из самых известных расширений сетей Петри являются так называемые *раскрашенные* сети Петри, обзору и применению которых для моделирования временных процессов посвящен следующий параграф.

При анализе сети Петри основное внимание уделяется, как правило, решению трех проблем.

- *Проблема достижимости*. В сети Петри с начальной разметкой M_0 требуется определить, достижима ли принципиально из M_0 некоторая разметка M_k. С точки зрения исследования моделируемой системы эта проблема интерпретируется как проблема достижимости (реализуемости) некоторого состояния системы.
- *Оценка живости переходов сети*. Под живостью перехода T_j понимают возможность его срабатывания в данной сети при начальной разметке M_0. Анализ модели на свойство живости позволяет выявить невозможные состояния в моделируемой системе (например, неисполняемые ветви в программе).
- *Оценка безопасности сети*. Безопасной является такая сеть Петри, в которой ни при каких условиях не может появиться более одной метки в каждой из позиций. Для исследуемой системы

это означает возможность функционирования ее в стационарном режиме. На основе анализа данного свойства могут быть, например, определены требования к буферным накопителям в системе.

Пример 2.3. Моделирование процесса интерактивного обучения [15, 16].

Применим рассмотренную выше теорию к моделированию процесса обучения. Рассмотрим модель прохождения под управлением автоматизированной обучающей системы (АОС) интерактивного курса, содержащего ряд учебных модулей, которые должны быть изучены в последовательном порядке.

Процесс прохождения студентом учебного модуля заключается в следующем. Из базы учебных модулей извлекается очередная порция теоретического материала, которую предлагается освоить обучаемому. После того как обучаемый окончил изучение этого материала, система приступает к тестированию. Из базы тестов выбирается тестовый материал и предъявляется обучаемому, который готовит и вводит в систему ответы на тестовые задания. Эти ответы анализируются системой оценивания, которая принимает решение:

- ответы верные: в этом случае изучение данного модуля завершается и возможен переход к следующему модулю;
- ответы неточные: в этом случае обучаемый должен изучить дополнительный материал и затем пройти повторное тестирование;
- ответы абсурдные: в этом случае обучаемый должен изучить материал модуля с самого начала.

Построение модели будет вестись на основе принципа «от простого к сложному» для того, чтобы продемонстрировать моделирующие возможности сетей Петри и в то же время облегчить понимание используемых формализмов. Модель этого процесса в виде сети Петри представлена на рис. 2.6.

Эта сеть, в соответствии с приведенным выше описанием, содержит два множества узлов: множество позиций $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$ (обозначены кружками) и множество переходов $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$ (обозначены планками). Узлы соединены ветвями двух видов: от позиций к переходам и от переходов к позициям. Рассмотрим смысл условий и событий, происходящих в системе.

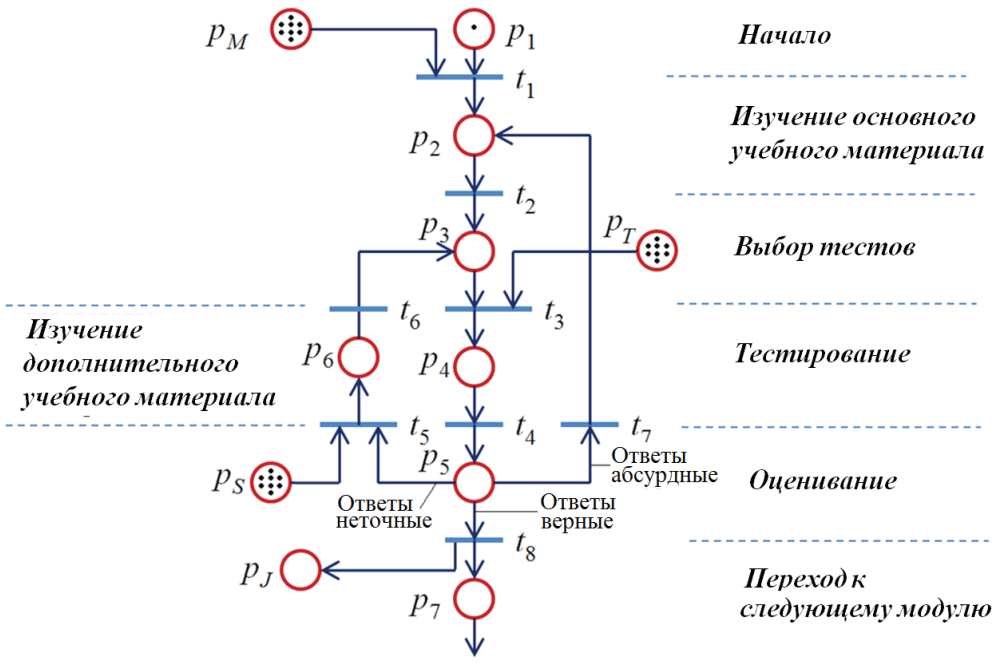


Рис. 2.6. Сеть Петри, моделирующая изучение отдельного модуля

Условия, моделируемые позициями:
p_1 — изучение модуля возможно;
p_2 — основной материал модуля выбран;
p_3 — выбор теста возможен;
p_4 — тест выбран;
p_5 — оценивание ответа произведено;
p_6 — дополнительный материал модуля выбран;
p_7 — переход к следующему модулю возможен;
p_M — база основных учебных модулей;
p_S — база дополнительных материалов;
p_T — база тестовых материалов;
p_J — журнал учета пройденных модулей.
События, моделируемые переходами:
t_1 — изучение основного материала модуля начинается;
t_2 — изучение основного материала модуля завершается;
t_3 — тестирование начинается;
t_4 — тестирование завершается;
t_5 — изучение дополнительного материала начинается;

t_6 — изучение дополнительного материала завершается;

t_7 — повторное изучение модуля начинается;

t_8 — изучение модуля завершается.

Маркировка позиций моделирует выполнение условий, а переходы при своем срабатывании — наступление событий. Смысл введенного ресурса следующий: если в позиции p_i имеется хотя бы одна фишка (т. е. маркировка $m_i = k's,\ k > 0$), то срабатывание выходного перехода возможно. Описанная сеть называется примитивной (обыкновенной) сетью Петри, с изучения таких сетей началась разработка рассматриваемой теории.

В этом случае выражения на всех ветвях имеют один и тот же вид $1's$, что означает передачу по ветви единичного целочисленного ресурса. Поэтому на рисунке выражения на ветвях не показаны.

Начальная маркировка позиций, как показано на рисунке, выглядит следующим образом:

$$m_1 = 1's,\ m_M = M's,\ m_S = N's,\ m_T = L's. \qquad (2.7)$$

Здесь M — количество модулей в курсе, N — количество дополнительных разделов, L — количество тестов. Предполагается, что $M < N << L$. Все остальные позиции в начальный момент не содержат ресурсов, т. е. имеют нулевую маркировку.

Поясним работу сети. В соответствии с правилами функционирования сети Петри (2.6) на первом шаге может сработать переход t_1 (что соответствует событию: «Изучение основного материала модуля начинается»). При этом будет изъято по одной фишке из позиций p_M и p_1 и одна фишка будет помещена в позицию p_2. Выполнится соответствующее условие и появится возможность сработать переходу t_2. Описанный процесс продолжится, аналогичным образом сработают переходы t_2, t_3 и t_4. После выполнения условия p_5 «Оценивание ответа произведено» возможно разветвление процесса по трем направлениям, которые моделируются переходами t_5, t_7 и t_8. Условие p_5 на самом деле представляет собой иерархическую позицию (см. рис. 2.5), в которой должен быть реализован оператор множественного выбора. При срабатывании перехода t_8 и попадании фишки $1's$ в позицию p_7 моделируемый процесс завершается. Кроме того, помещается фишка $1's$ в позицию p_J, что соответствует фиксации этого факта в журнале учета АОС.

2.2. Раскрашенные сети Петри

Методология моделирования динамики дискретных систем, основанная на формализме раскрашенных (цветных) сетей Петри — *Coloured Petri Net* (*CPN*), оказалась эффективным средством моделирования широкого круга динамических систем, состоящих из независимо работающих, но взаимодействующих элементов, в том числе бизнес-процессов.

Особенность этой методологии состоит в том, что она моделирует системы в терминах «*разнотипные* условия — события», что позволяет исследовать динамику работы сложной системы. Модели на основе *CPN*-схем («событийные модели») в настоящее время встраиваются в универсальные системы моделирования информационных систем, таких как ARJS, IDEF и другие. Для автономного использования методологии *CPN*-схем разработан специальный язык моделирования *Coloured Petri Net Modeling Language* (*CPN ML*) [17] и созданы соответствующие программные средства. С возможностями системы (*CPN/Tools*), работающей под *Windows*, можно ознакомиться на сайте http://www.daimi.au.dk/CPnets/CPN2000.

Методология *CPN* близка к структурным методам моделирования систем, однако в отличие от многих из них она базируется на хорошо разработанном математическом аппарате и поэтому допускает проведение аналитических и имитационных исследований. Полное и строгое описание *CPN* содержится в трехтомной монографии Курта Йенсена [18, 19, 20]. Кроме того, для практического использования временных сетей Петри и в телекоммуникации могут оказаться полезными монографии [21, 22].

Как следует из названия, модель динамической системы в методологии *CPN* представляет собой сеть — двудольный ориентированный мультиграф. Сеть, как и ранее, содержит узлы двух видов — позиции и переходы, связанные ветвями. При этом позиции моделируют наличие в системе определенных ресурсов, необходимых для наступления события. Различные виды ресурсов условно обозначаются разными «цветами», что объясняет название методологии. Переходы моделируют сами события. Наступление какого-либо события в системе изменяет условия, что может привести к новым событиям. Ветви моделируют причинно-следственные связи в системе, а развернутая во времени последовательность условий и событий во времени образует модель динамики системы.

2.2.1. Формальное определение *CPN* [11, 12]

Раскрашенная сеть Петри *CPN* — это кортеж, состоящий из восьми элементов:

$$CPN =< P, I, T, G, A, E, Z, C >. \quad (2.8)$$

Рассмотрим элементы определения (2.8).

1. P — конечное множество позиций. С каждой позицией может быть связана определенная маркировка, которая учитывает наличие в данной позиции различных типов ресурсов. Маркировка позиции $p \in P$ представляет собой мультимножество, например, следующего вида: $m(p) = (1'r, 2'b, 1'g)$. Здесь r, b, g — переменные указанных цветовых типов, определяющие различные виды ресурсов, а цифры, стоящие перед апострофами, — количество фишек соответствующего типа в позиции $p \in P$.

Разметка раскрашенной сети $M = [m(p_1), m(p_1), \ldots, m(p_n)]$, т. е. массив мультимножеств, определяющих разметку позиции, может быть представлен в системе MATLAB структурой вида

```
m(1) = struct('r',1,'b',2,'g',1)
m =
    r: 1
    b: 2
    g: 1
```

Операции над мультимножествами можно определить соответствующими процедурами, которые должны быть едиными для всех позиций.

2. $I(p)$ — функция инициализации сети *CPN*. Эта функция по аналогии с обыкновенными сетями Петри задает начальную разметку сети M_0, т. е. для каждой позиции $p \in P$ указывает цветовое мультимножество, обозначаемое $m_0(p)$, например, $m_0(p_1) = 3'e$; $m_0(p_2) = 2'(z, 0)$; $m_0(p_3) = \varnothing$ — начальные разметки позиций. Например, в системе MATLAB начальная разметка сети $M_0 = \{3'e;\ 2'(z, 0);\ \varnothing\}$ может быть представлена в виде массива ячеек.

3. T — конечное множество переходов, которое можно представить перечислимым типом $T = [1;\ 2; \ldots; m]$ или одномерным массивом.

4. $G(t)$ — блокировочная (спусковая) функция, описывающая дополнительные условия, которые должны быть выполнены для срабатывания перехода $t \in T$. Эта функция представляет собой предикат, со-

ставленный из переменных, принадлежащих типам цветов Z. Например, логическая функция

$$G(t) = \begin{cases} x = g, \ if \ t = t_1; \\ true, otherwise, \end{cases}$$

```
function y = G(t,p)
y = true;
if nargin > 1
   if t(1)
      y = (p(1) == p(2)); end
end
```

принимает истинное значение (срабатывание) для всех переходов, кроме t_1, для которого должно выполняться условие $x = g$, в других случаях она ложь (справа ее перевод на язык системы MATLAB, `p` — масив возможных параметров функкции). Таким образом, для того чтобы было разрешено срабатывание перехода t_1, требуется выполнение дополнительного условия $x = g$ на входной ветви к t_1. Если эта функция не определена, то по умолчанию предполагается истинной.

5. A — конечное множество ветвей, связывающих между собой позиции и переходы. В языке *CPN ML* указываются все ветви с помощью выражений вида: p to t и t to p, где $p \in P, t \in T$. Такая нотация введена для того, чтобы пару элементов (p_i, t_j) можно было соединить несколькими ветвями с различными свойствами. В этих выражениях первый элемент указывает начало ветви, второй элемент — конец ветви.

Множество A имеет структуру $A = [a_1, a_2, \ldots, a_l]$, где ветви $a_k \in A$ определяются одним из видов выражений: $a_k = (p_i, t_j)$ или $a_k = (t_j, p_i)$, $p_i \in P$, $t_j \in T$. Поскольку ветви имеют два вида: от позиции к переходу (p to t) и от перехода к позиции (t to p), то множество A можно представить в виде суммы непересекающихся множеств: $A = A_p \cup A_t$, $A_p \cap A_t = \varnothing$, где множество A_p содержит элементы вида (p to t), а множество A_t — элементы вида (t to p). Элементы множеств P, T и A не пересекаются: $P \cap T = P \cap A = T \cap A = \varnothing$.

6. $E(a)$ — функция, задающая выражения на ветвях $a \in A$. Эта функция для каждой ветви a определяет мультимножество, состоящее из элементов, описанных в множестве цветов Z. Мы будем говорить, что это мультимножество помечает ветвь. Введение в *CPN* функции $E(a)$ является развитием понятия кратности ветвей в формализме обыкновенных сетей Петри.

Рассмотрим примеры задания функции $E(a)$.

$E(a_1) = 2'r$ — ветвь a_1, помеченная двумя метками типа r;

$E(a_2) = case\ \ x\ \ of\ \ \ p \Rightarrow 2'r \,|\, q \Rightarrow 1'r$ — ветвь a_2 помечается двумя метками типа r, если переменная $x = p$, и одной меткой типа r, если $x = q$;

$E(a_3) = if\ \ x = p\ \ then\ 1'r\ \ else\ \ empty$ — ветвь a_3 помечается одной меткой типа r, если $x = p$, иначе не помечается;

$E(a_4) = if\ \ x = q\ \ then\ 1'(q, i+1)\ \ else\ \ empty$ — ветвь a_4 помечается одной меткой типа R, если $x = q$, и не помечается в противном случае.

Отсутствие выражения для какой-либо ветви означает, что ветвь не помечена.

7. Z — конечное множество непустых типов, называемое цветами. Типы имеют общее название `color` и задаются в системе MATLAB в виде массивов ячеек.

Пример 2.4. Формирование массива цветных мультиметок.

Создадим сначала массив `color1` из трех цветных меток `r`, `g`, `b`, затем второй массив `color2` из трех целых меток `1, 2, 3` и сформируем прямое произведение этих массивов для получения массива `col3` цветных мультиметок.

```
color1={'r'; 'g'; 'b'};
color2={1; 2; 3};
for i=1:3, for j=1:3
color3{i, j}= [' (' num2str (color2{i}) ',' color1{j} ')'];
end, end
```

Получившийся массив ячеек `colorl3` состоит из 9 пар элементов:

```
{'(1,r)'   '(1,g)'       '(1,b)';
 '(2,r)'   '(2,g)'       '(2,b)';
 '(3,r)'   '(3,g)'       '(3,b)'}
```

8. $C(p)$ — цветовая функция, определяющая множество типов цветов, разрешенных для каждой позиции. Например, запись

$$C(p) = \begin{cases} color1, & if\ \ p \in \{p_1, p_2, p_3\}; \\ color2, & otherwise, \end{cases}$$

означает, что цвета типа $color1$ разрешены для позиций p_1, p_2, p_3, а цвета типа $color2$ разрешены для всех остальных позиций из P.

2.2.2. Раскрашенные сети Петри с временным механизмом

Существует ряд задач моделирования, в которых необходимо учитывать не только последовательность событий, но и время их наступления, а также продолжительность. Для этой цели предусмотрено расширение возможностей раскрашенных сетей Петри путем введения временного механизма (так называемых *timed CPN* [11] или *tCPN*). В несколько упрощенном виде сущность такого расширения описана ниже:

$$tCPN = < P, I, T, G, A, E, Z, C, \tau, \Delta t >. \qquad (2.9)$$

В модель системы вводятся часы, показывающие глобальное время τ. Обычно это время считается дискретным, т. е. означает номер такта, выдаваемого тактовым генератором системы моделирования. Глобальное время τ отличается от времени k, которое содержится в определении (2.6), поскольку k есть номер шага работы *CPN*, а τ изменяется независимо от работы сети.

Ресурсы, перемещаемые в сети (фишки), могут получить временные метки. Такие ресурсы в общем виде задаются мультимножествами с временными метками (*timed multi-sets*), однако мы эту теорию не рассматриваем. Отметим лишь, что при описании множества цветов добавляются пометки *timed*, а переменные соответствующего типа снабжаются знаками @ (по-английски читается *at*, т. е. «во время»). Это означает, что переменная привязана к глобальному времени. После значка @ в квадратных скобках указывается значение глобального времени, в течение которого возможно использование данных фишек для срабатывания переходов, для которых они являются входными. При этом запись вида @ [500] говорит о том, что фишка «включается» в момент $\tau = 500$ и далее готова для работы в сети, а запись @ [500, 600] означает, что фишка может использоваться в диапазоне глобального времени $500 \le \tau \le 600$.

Пример 2.5. Формирование массива цветных меток с временем активации.

Выполним сначала формирование массива **`col3`** цветных мультиметок (как в примере 2.2), затем сформируем массив **`col4`**, в котором создадим два поля: одно — **`metka`** — обозначает цветные мультиметки, второе — **`timed`** — время их активации.

```
col1 = {'red'; 'green'; 'blue'};  % Массив из трёх цветных меток
col2 = {1; 2; 3; 4};              % Массив из четырёх целых меток
% Создается массив col3 из 12 мультиметок
for i=1:4,  for j=1:3
   col3{i,j} = ['('  num2str(col2{i})  ',' col1{j} ')'];
end,  end
% Создается 12-элементный массив структур col4 с полем metka
col4 = cell2struct(col3(:)',{'metka'},1);
% В массив структур col4 добавляется временное поле timed,
% а его первому элементу присваивается значение времени 12
col4(1).timed = 12; % 12-элементный массив из двух полей
```

В итоге получен массив **col4** цветных мультиметок с временными свойствами:

```
12x1 struct array with fields:
    metka
    timed
```

В этом случае **col4** представляет собой массив структур, состоящих из двух полей — **metka** и **timed**, и, например, первые два элемента массива структур будут иметь значения:

```
col4 (1) → metka: ' (1, red)'
timed: 12
```

```
col4 (2) → metka: ' (1, red)'
timed: []
```

Например, возможное значение мультимножества, определяемого переменной x: 2'(1,'blue') @ [12], означает, что переменная x содержит две метки (два мультимножества), каждая из которых содержит два ресурса: целочисленный ресурс, равный 1 из множества **col2**, и цвет **'blue'** из цветового множества **col1.** Эта переменная «включается» в момент $\tau = 12$.

Каждый переход, на вход которого поступают метки, имеющие временные свойства, получает дополнительное условие срабатывания: переход может сработать только в том случае, если системное время удовлетворяет всем условиям на входных метках. В свою очередь, при срабатывании переход может увеличить временное свойство метки, т. е. смоделировать задержку в работе системы. Величина задержки определяется специальным выражением, связанным с переходом и имеющим вид $@+\Delta t$, где Δt — время задержки, задаваемое числом или функцией.

Пусть в сети на рис. 2.7 начальная разметка такова: $m_1 = 2'(1,'\text{blue}') @ [12]$, $m_2 = \varnothing$. Выражения на ветвях показаны на *tCPN*-схеме. Глобальное время $\tau = 12$. Переход t может сработать, при этом он осуществит задержку передачи меток в p_2 на 2 единицы времени в соответствии с выражением на t. Таким образом, после срабатывания t получим разметку: $m_1 = \varnothing$, $m_2 = 2'(1,'\text{blue}') @ [14]$.

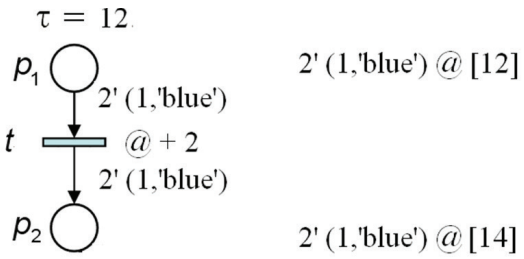


Рис. 2.7. Фрагмент сети Петри с временными метками

Пример 2.6. Многомодульная модель прохождения учебного курса [15].

В этой сети (см. рис. 2.8) рассмотренная выше модель модуля интерактивного обучения (см. пример 2.3) представлена в виде одного составного перехода, моделирующего так называемое непримитивное событие — прохождение учебного модуля.

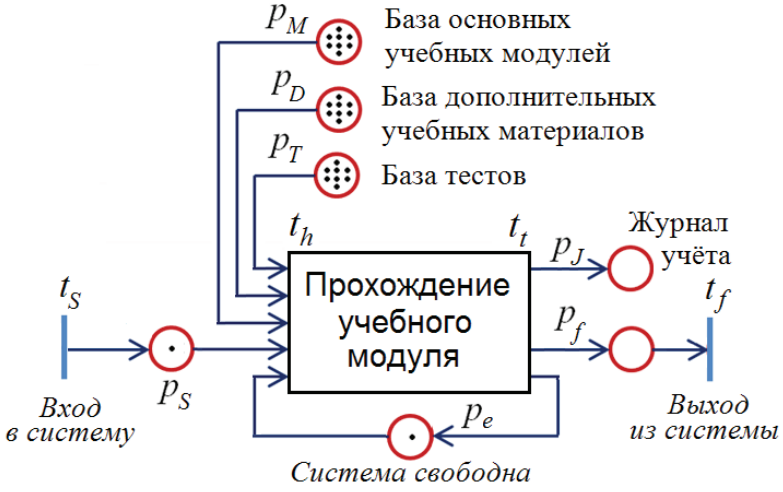


Рис. 2.8. Сеть Петри, моделирующая прохождение учебного курса

Непримитивное событие в нотации сетей Петри имеет приблизительно тот же смысл, что подпрограмма в языках программирования. Такое событие изображается прямоугольником, боковые стороны которого представляют собой входной и выходной переходы t_h и t_t. События, моделируемые этими переходами, следующие:

t_h — изучение очередного модуля начинается;

t_t — изучение очередного модуля завершается.

Помимо упомянутых уже позиций p_M, p_D, p_T, p_J, моделирующих внешние условия по отношению к учебному модулю, в системе присутствуют еще три позиции:

p_S — обучаемый присутствует,

p_e — обучающая система свободна,

p_f — курс пройден,

и два перехода:

t_s — обучаемый входит в систему,

t_f — обучаемый выходит их системы.

Функционирование системы происходит при начальной маркировке:

$$m_0 = 1's, m_e = 1's, m_M = N's, m_S = K's, m_T = L's. \qquad (2.10)$$

При начале работы очередного модуля из позиции p_M извлекается одна фишка, затем извлекаются фишки из позиции p_T и, возможно, из p_D. По окончании изучения очередного модуля одна фишка помещается в позицию p_J.

Процесс продолжается до тех пор, пока в позиции p_M остается хотя бы одна фишка. В том случае, когда позиция p_M опустеет, в позиции p_J оказывается N фишек, а в позициях p_f и p_e — по одной. Таким образом создаются условия для удаления из системы учащегося, окончившего курс (срабатывание перехода t_f), и для начала обучения нового учащегося (срабатывание переходов t_0 и t_h). При этом должна обновиться начальная маркировка сети на рис. 2.8.

Приведенная выше одноресурсная модель не отражает ряда существенных особенностей процесса прохождения курса:

- не учитывается вероятностный характер этого процесса;
- не фиксируется время, затрачиваемое на обучение, доучивание и тестирование;
- не учитываются номера и порядок прохождения учебных модулей, привязка дополнительного материала и тестов к номерам модулей.

Использование формализма раскрашенной сети Петри позволяет в определенной мере устранить отмеченные выше особенности. На рис. 2.9 показана временная CPN, моделирующая процесс изучения курса, близкий к реальному. Она получена путем замены составного перехода на рис. 2.8 на сеть, изображенную на рис. 2.6, с некоторыми незначительными корректировками мест подключения ветвей к переходам и позициям.

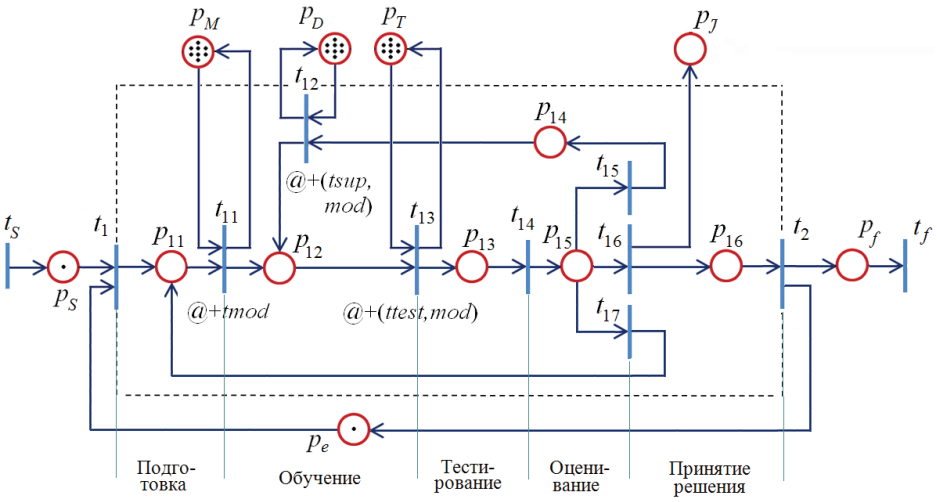


Рис. 2.9. Раскрашенная сеть Петри с временным и вероятностным формализмами, моделирующая прохождение учебного курса

Выражения на переходах t_{11}, t_{12}, t_{13}, показанные на рис. 2.9, обозначают временную задержку срабатывания, т. е. время выполнения обучаемым соответственно операций изучения основного модуля, изучения дополнительного материала и тестирования. По сравнению с рис. 2.6 добавлены ветви обратной связи, моделирующие процесс возврата модулей $\{1'1 \ldots 1'M\}$ в хранилище p_M, дополнительных материалов $\{1'(1,1) \ldots 1'(M,N)\}$ в хранилище p_D, тестовых материалов $\{1'(1,1) \ldots 1'(M,L)\}$ в хранилище p_T после их использования.

Порядок функционирования раскрашенной сети CPN на рис. 2.9 почти не отличается от рассмотренного выше для одноресурсной сети на рис. 2.6. При указанной выше начальной маркировке может сработать переход t_1, после чего последовательно работают переходы t_{11}, t_{13}, t_{14}. Затем в зависимости от полученного случайным образом значения оценки (переменной (*rat, mod*)) срабатывают переходы t_{15}, t_{16}, t_{17}.

В первом случае (оценка равна 3) срабатывает переход t_{15} и создаются условия для срабатывания перехода t_{12}, что соответствует изучению дополнительного материала. Во втором случае (оценка больше 3) срабатывает переход t_{16} и, во-первых, направляется запись в журнал о завершении изучения очередного модуля (в позицию p_J направляется мультимножество $1'mod+1'test+1's$), а, во-вторых, осуществляется проверка: все ли модули пройдены. Если пройден последний модуль ($mod == M$), то при срабатывании перехода t_2 очередной обучаемый с номером s удаляется из системы и дается разрешение на вход в систему следующего обучаемого с номером $s+1$.

Этот пример показывает, что временная CPN дает возможность провести более детальное моделирование учебного процесса: дифференциация ресурсов позволяет проследить последовательность прохождения модулей, количество возвратов на дополнительное и повторное изучение материала; введение временного формализма — проследить время, затраченное учащимся на выполнение отдельных операций, и составлять протоколы прохождения курса. Дополнительное использование специализированных программных средств позволяет помимо имитационного моделирования проводить разнообразные исследования свойств процесса.

Можно построить полное *дерево маркировок* сети при заданной начальной маркировке и выявить особые ситуации, например, тупиковые маркировки, когда не может сработать ни один переход, и система оказывается заблокированной. Для сложных систем такая задача часто является весьма актуальной, трудно поддающейся анализу из-за большого числа возможных комбинаций состояний системы. На основе анализа дерева маркировок решается также задача *достижимости* определенного состояния системы и выявления последовательности событий, приводящих систему в данное состояние. Во-вторых, возможно исследование инвариантов системы, позволяющих оценить баланс ресурсов в системе и наличие циклов смены состояний.

2.2.3. Вложенные сети Петри

Во вложенных сетях Петри фишки, помечающие позиции, рассматриваются как объекты, имеющие самостоятельное поведение, которое описывается также некоторыми сетями Петри. Название «вложен-

ные сети» указывает на то, что элементы сетей в них сами являются сетями, подобно тому, как в системе вложенных множеств элементами некоторого множества могут быть другие множества.

Вложенная сеть Петри состоит из системной (корневой) сети и множества элементных (парциальных) сетей, представляющих фишки системной сети. Поведение вложенной сети Петри включает четыре типа шагов.

Системный шаг переноса — это срабатывание перехода системной сети в соответствии с обычными правилами для сетей Петри высокого уровня, при этом элементные сети рассматриваются как фишки, не имеющие собственной структуры.

Элементно-автономный шаг меняет только внутреннее состояние (маркировку) элементной сети, не меняя ее местонахождения в системной сети.

Шаг горизонтальной синхронизации означает одновременное срабатывание двух переходов в двух элементных сетях, находящихся в одной позиции системной сети (эти переходы должны быть помечены специальными метками горизонтальной синхронизации).

Шаг вертикальной синхронизации используется для синхронизации перехода в системной сети с некоторыми переходами элементных сетей. Вертикальная синхронизация означает одновременное срабатывание перехода системной сети и переходов (помеченных дополнительной меткой) в задействованных в этом срабатывании элементных сетях.

Вложенные сети Петри обладают рядом преимуществ, которые делают их удобным инструментом для моделирования и анализа алгоритмов управления сложными динамическими системами: иерархическая и модульная структура, присутствие элементарных сетей с собственным строением и поведением, наличие механизмов горизонтальной и вертикальной синхронизации, динамическое распараллеливание общей задачи.

Например, функциональность рассмотренной выше в примере 2.6 модели интерактивного учебного процесса можно повысить, если поведение каждого обучаемого описывать с помощью отдельной сети Петри. Тогда фишка, обозначаемая переменной *s* на входе сети на рис. 2.9, станет вложенной сетью *ENs* (*Enclosed Network*), где *s* — цифровой код обучаемого. При этом получится вложенная сеть Петри (*Nested Color Petri Net* = *NCPN*), которая состоит из системной сети *SN* и набора элементных сетей *ENs* ($s = 1, 2, ...$). Детальное описание одного из возможных вариантов сети *ENs* представлено в [16].

В заключение пункта приведем блок-схему процедуры разработки модели сложной ситемы с наличием большого числа причинно-следственных связей (рис. 2.10). При ее построении используется комплексная методология системного и классического подходов:

1) сначала в результате декомпозиции системы создается банк CPN-моделей, при этом декомпозция системы на отдельные компоненты проводится согласно поставленной цели моделирования;

2) CPN-модели соединяются между собой с учетом свойств (цветов) и временных задержек соответствующих компонентов, проводится многократная подстройка (адаптация) выбранных моделей и их разделение на основные, дополнительные и сопутствующие;

3) далее согласно системным связям из классифицированных адаптивных CPN-моделей формируется компьютерная модель, ориентированная на использование определенного программного инструмента;

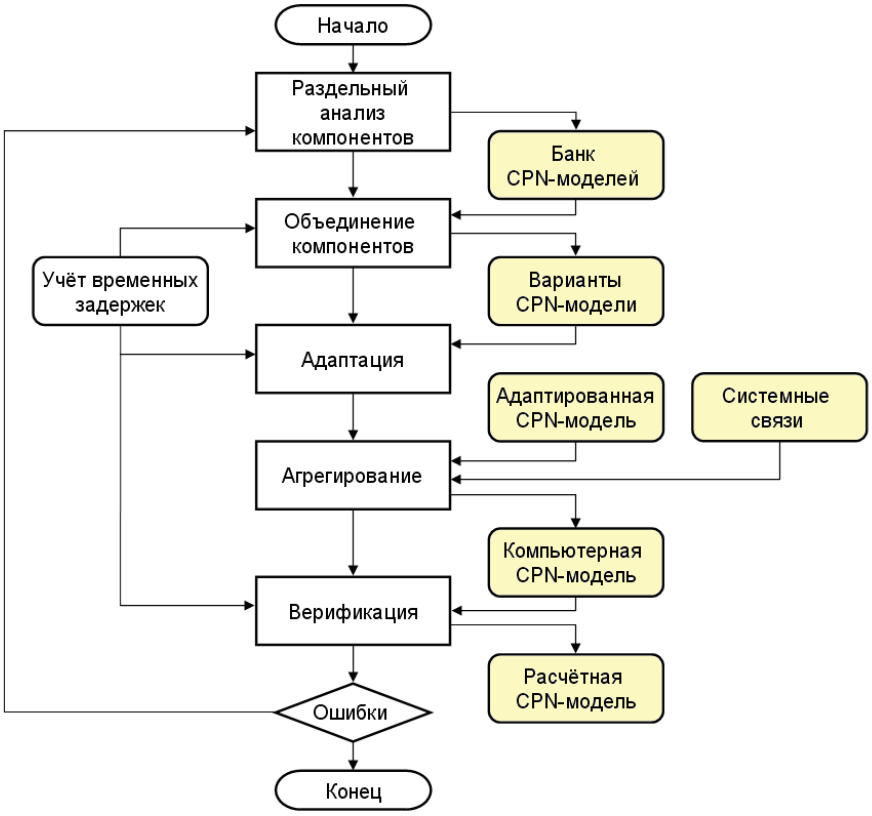


Рис. 2.10. Блок-схема алгоритма исследования систем с использованием CPN-сетей

4) наконец, по результатам прогонки компьютерной модели системы на основе принятого показателя качества принимается решение об адекватности модели: если ошибок нет или они меньше допустимых, то CPN-модель признается пригодной для расчетов (моделирования), в противном случае процесс разработки возвращается на более ранние стадии.

Примером приложения указанной выше методики может быть разработка функциональной модели информационного портала сетями Петри, апробированная созданием на ее основе нескольких реальных порталов научно-образовательной тематики [23].

Теория сетей Петри развивается в нескольких направлениях: разработка математических основ, структурная теория сетей, различные приложения (параллельное программирование, дискретные динамические системы и т. д.).

2.3. Инструменты моделирования CPN

В параграфе представлено краткое описание возможностей одной из популярных моделирующих систем — программы CPN Tools, разработанной в университете Орхуса (Дания) для моделирования телекоммуникационных систем и сетей [17]. Языком построения и анализа математических моделей являются раскрашенные сети Петри. Программа является новым поколением ранее использованной Design-CPN и в настоящее время реализована в ОС Windows и на платформах Unix.

Программа CPN Tools может применяться для моделирования систем управления производственными системами, транспортными средствами, планирования военных операций. В области телекоммуникаций она часто используется при проектировании телекоммуникационных устройств и сетей, оценки их пропускной способности, для спецификации и верификации протоколов. Полный список реальных применений можно найти на домашней странице CPN Tools http://www.daimi.au.dk/CPNTools/.

2.3.1. Основные функции CPN Tools

Основными функциями программы CPN Tools являются:

- разработка и редактирование моделей на основе сетей Петри;
- анализ поведения моделей с помощью имитации динамики сети Петри;
- построение и анализ пространства состояний модели.

Для создания моделей предусмотрен специальный графический редактор раскрашенных сетей Петри. Редактор позволяет рисовать сети Петри на экране компьютера, вводить атрибуты элементов сети и дополнительные описания на встроенном языке CPN ML. Модель может состоять из нескольких страниц, которые связаны друг с другом и образуют иерархические структуры.

Для достаточно простых моделей возможна генерация полного пространства состояний (*графа достижимости*). Это — лучший способ для верификации, например, телекоммуникационных протоколов. CPN Tools обеспечивает построение пространства состояний и автоматическую генерацию по нему отчета, который содержит выводы о стандартных свойствах сетей Петри, таких как *ограниченность* и *живучесть*. Кроме того, предусмотрен специальный язык на основе языка CPN ML для описания запросов о нестандартных свойствах пространства состояний, которые важны для пользователя. К сожалению, для сложных моделей пространство состояний может быть слишком большим, и его построение не представляется возможным.

Единственный способ для анализа сложных моделей — это имитация их поведения. CPN Tools предусматривает пошаговую имитацию для поиска и устранения ошибок в разрабатываемой модели, а также автоматическое выполнение определенного количества шагов. Имитация на больших временных интервалах — это путь для статистического анализа поведения модели. Такой подход применяется для оценки характеристик телекоммуникационных сетей, например, пропускной способности и качества обслуживания.

2.3.2. Основы языка CPN ML

Встроенный язык CPN ML используется для создания описаний в индексе и надписей элементов сетей (атрибутов) и обеспечивает описание множеств цветов (типов данных), переменных, функций, ве-

личин (констант). У каждой позиции раскрашенной сети Петри должен быть обязательный атрибут — определенное множество цветов, и она может содержать фишки только указанного множества цветов. Переменные и функции используются для указания атрибутов переходов и ветвей.

Описания расположены в индексе как части сети. Существуют стандартные предопределенные описания таких множеств цветов, как **E** — элементарный тип, **INT** — целые числа, **BOOL** — логический тип, **STRING** — строковый тип. Описания пользователя могут быть добавлены после стандартных описаний, с помощью контекстно-зависимых меню. Кроме того, для сложных сетей CPN Tools обеспечивает внешние описания, которые могут быть загружены из файла.

Составные множества цветов состоят из комбинаций простых множеств цветов. CPN ML обеспечивает следующие составные множества цветов: **product**, **record**, **union**, **list**, **subset** и **alias**. Множества **product** и **record** представляют собой кортежи данных, сформированные как результат декартового произведения множеств цветов. Единственное различие между ними состоит в том, что компоненты множества цветов **product** являются безымянными, в то время как компоненты множества цветов **record** поименованы. Существует близкое сходство с типом данных **record** языка программирования Паскаль или со структурами языка C. Множества **list** и **union** редко используются и являются более сложными.

Пример 2.7. Описание кадра Ethernet на языке CPN ML.

Кадр Ethernet состоит из адреса отправителя, адреса получателя и данных. Представим MAC-адреса множеством цветов типа **integer** и данные — множеством цветов типа **string**:

```
colset MAC = int;
colset DATA = string;
colset frame = product MAC * MAC * DATA;
colset frame1 = record src : MAC * dst : MAC * d : DATA;
```

Фреймы Ethernet могут быть представлены множеством цветов **frame** либо множеством цветов **frame1**. Для множества цветов **frame** переменная **x = (2, 4, «Hello»)**, например, описывает кадр, отправленный устройством 2 на устройство 4, содержащий приветствие «Hello». Этот же кадр для множества цветов **frame1** имеет вид **x1 = {dst=4, src=2, d = «Hello»}**. В последнем случае принадлежность адресов и данных

строго связана с именем параметра («цветом» метки), поэтому в отличие от фрейма **frame** порядок описания кадра Ethernet внутри фрейма **frame1** может быть произвольным.

В языке CPN ML применяются стандартные управляющие структуры универсальных языков программирования, такие как операторы **if**, **if-then-else** и **case**:

```
if bool-exp then exp1 else exp2;
```

где **exp1** и **exp2** имеют тот же самый тип;

```
case exp of
pat1 => exp1
| pat2 => exp2
| …
| patn
```

Синтаксис описания функции имеет вид:

```
fun id pat1 = exp1
| id pat2 = exp2
| …
| id patn = expn;
```

где **pat1**, **pat2**, ..., **patn** — шаблоны и выражения **exp1**, **exp2**, ..., **expn** имеют один и тот же тип. Описание означает, что в случае, когда фактические аргументы удовлетворяют шаблону **pati**, значение функции будет вычислено как **expi**. Но так как ML представляет собой, по существу, язык функционального программирования, то наибольшая изобразительная мощность достигается при создании рекурсивных функций. Например, следующая функция вычисляет факториал целого числа, используя рекурсию:

```
fun fact (0) = 1 | fact (i) = i * fact (i-1);
```

Пример 2.8. Модель передачи информационных пакетов по сети передачи данных.

Пример взят из [19] (Sect. 5.5) и описывает в среде CPN Tools простой протокол, которым отправитель может передать конечный набор информационных пакетов приемнику. Канал передачи данных может потерять пакеты или пакеты могут прийти к приемнику не в порядке их посылки. Поэтому может появиться необходимость повторно передать пакеты или игнорировать копии и пакеты, которые появляются не в порядке посылки. Хотя пример описывает довольно простой

протокол, однако он достаточно представителен, чтобы быть интересным для более близкого исследования.

CPN-модель протокола показана на рис. 2.11. Она состоит из трех частей: отправителя данных, канала передачи данных и приемника данных. Отправитель Sender представлен двумя переходами, которые могут послать пакеты и получить подтверждения об их приеме. Канал передачи Network также имеет два перехода, которые могут передать сами пакеты и сигналы подтверждения об их приеме. Наконец, приемник Receiver имеет единственный переход, который может получать пакеты (и послать сигнал подтверждения). Интерфейс между отправителем и сетью состоит из позиций A и D, в то время как интерфейс между сетью и приемником состоит из позиций B и C.

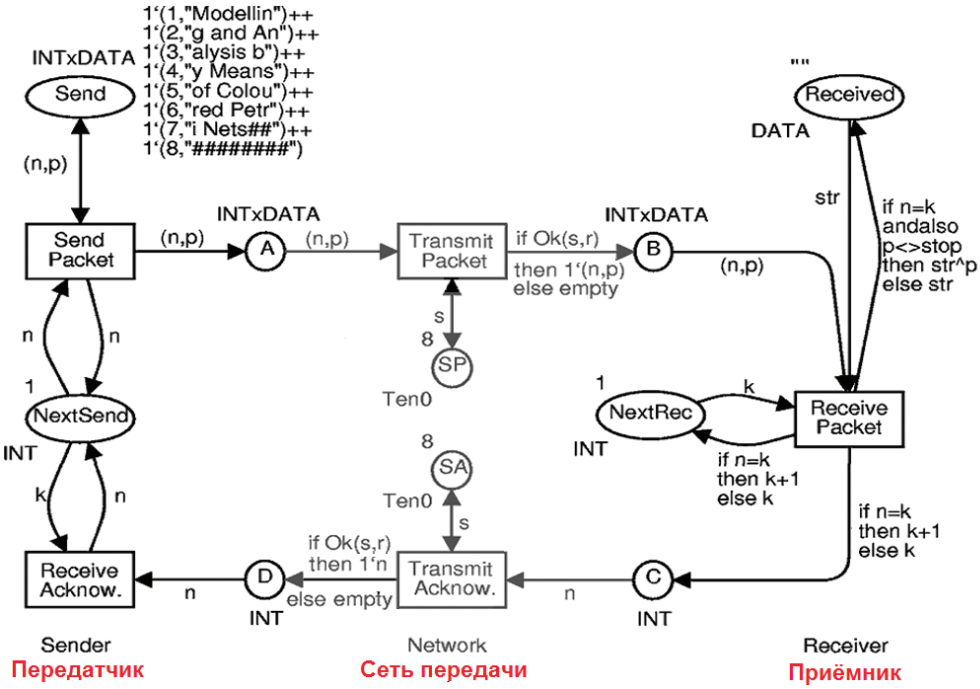


Рис. 2.11. CPN-модель пакетной передачи данных

Передаваемые пакеты расположены на рис. 2.11 около позиции Send в верхнем левом углу. Каждая метка этой позиции содержит номер пакета и его содержание — данные пакета, представленные как последовательность символов текста. Позиция NextSend содержит номер следующего пакета, который будет послан. Первоначаль-

но это число равно 1, и оно обновляется каждый раз, когда получено подтверждение.

Содержание полученного сообщения сохраняется в позиции Received (в верхнем правом углу рисунка). Эта позиция содержит единственную метку со строкой текста, которая составляется из подстрок текста, содержащихся в полученных пакетах (игнорируются содержания дубликатов и пакетов, полученных не в порядке). Первоначально строка текста в позиции Received пуста, т. е. равна « «. В конце передачи в позиции Received ожидается получить строку "Modelling and Analysis by Means of Coloured Petri Nets". Позиция NextRec содержит номер следующего пакета, который будет получен. Первоначально это число 1, и оно обновляется каждый раз, когда пакет успешно принят.

Модель не отображает того, как Sender преобразует полный текст сообщения в последовательность пакетов или как Receiver повторно собирает пакеты в сообщение. Для упрощения не моделируется также то, как метки в Send и Received удаляются в конце передачи или как номера пакетов в NextSend и NextRec повторно устанавливаются в 1.

Итак, в CPN-модели передачи данных присутствуют четыре содержательные позиции (Send, NextSend, Received и NextRec), две буферных позиции (SP и SA) и четыре интерфейсные (вспомогательные) позиции (A, B, C и D).

Также в CPN-модели имеется пять различных переходов. Первый переход SendPacket посылает пакет Network, создавая копию пакета в позиции A. Номер в NextSend определяет, который по порядку пакет послан. Следует отметить, что пакет не удаляется из позиции Send. Счетчик в NextSend не увеличивается. Причина в том, что пакет может быть потерян и, следовательно, должен быть повторно передан. Этот протокол является пессимистическим в том смысле, что продолжает повторять тот же самый пакет, пока не получит подтверждение о том, что пакет был успешно получен.

Второй переход TransmitPacket передает пакет от Sender к Network в сторону Receiver, перемещая соответствующую метку от позиции A до позиции B. Булево выражение *Ok* (*s*, *r*) определяет, передан ли пакет успешно или потерян. Переменная *r* должна быть привязана к произвольному значению в ее цветовом наборе (т. е. равна любому целому числу между 1 и 10). Функция *Ок* возвращает *true*, если значение *r* меньше или равно значению *s*. Это означает, что вероятность успешной передачи определяется меткой в буферной позиции SP. Посколь-

ку перед началом моделирования в SP задана метка со значением 8, то имеется 80 %-ный шанс для успешной передачи. Однако легко изменить величину успеха, просто изменив значение метки в SP.

Третий переход Receive Packet получает пакет и проверяет, идентично ли число пакета *n* числу *k* в позиции NextRec. Когда эти два числа совпадают, номер в NextRec увеличивается на 1 и подстрока текста передается им в позицию Received, где связывается с уже имеющейся строкой текста, если это не остановка = «########», которая в соответствии с соглашением указывает на конец сообщения. Иначе пакет игнорируется и число в NextRec остается неизменным. В обоих случаях посылается подтверждение, содержащее номер следующего пакета, который Sender должен послать.

Четвертый переход TransmitAcknow (Acknowledgment означает подтверждение) передает подтверждение от Sender к Network в сторону Sender, перемещая соответствующую метку от позиции C к позиции D. Работа перехода аналогичным образом моделирует переход TransmitPacket. Это означает, что подтверждение может быть потеряно с вероятностью, определенной меткой в позиции SA.

Пятый переход ReceiveAcknow, получив подтверждение на приемной стороне, обновляет число в позиции NextSend, заменяя старое значение полученным значением от позиции D.

После множества шагов CPN-сеть, возможно, достигла маркировки промежуточного звена, показанного на рис. 2.12. От левой части сети отправитель посылает пакет номер три. Три копии этого пакета присутствуют в позициях A и B. В правой части сети в Received была получена строка «Modelling and Analysis b». Это есть содержание первых трех пакетов, и приемник теперь ждет пакета номер четыре. Следовательно, пакеты на A и B будут игнорироваться, когда они достигают приемника. Мы можем также видеть, что два подтверждения присутствуют в позиции D. Когда ReceiveAcknow получает подтверждение Acknowledgment, состояние NextSend будет обновлено на четыре, и затем отправитель начнет посылать пакет номер четыре.

Когда последний пакет (с «########») успешно получен приемником, NextRec получает значение девять (на 1 больше, чем число пакетов). Это значение (через подтверждение) будет сообщено отправителю, NextSend обновлен на девять, и передача остановится, потому что никакого пакета с таким числом не существует. Еще после нескольких шагов, на которых позиции A, B, C и D будут очищены для паке-

тов/подтверждений, CPN-сеть достигнет заключительной (финальной) маркировки.

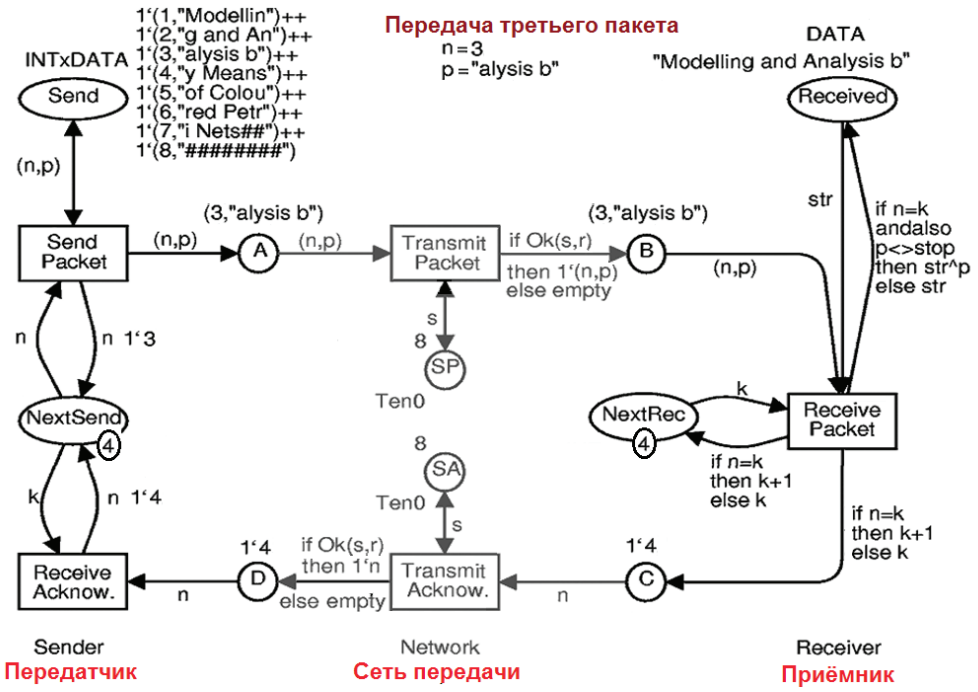


Рис. 2.12. Динамика движения пакетов в CPN-модели пакетной передачи данных

И хотя этот протокол довольно прост, но не так легко видеть, что эта его модель работает правильно. Что случится, например, если «последнее» подтверждение будет потеряно? Проведя ряд моделирований — в диалоговом или автоматическом режиме, пользователь сможет убедиться в правильности работы протокола. Он может также получить доказательство правильности при использовании графического инструмента.

Часто необходимо сделать запись событий в течение моделирования, например, когда выполняется прямое моделирование, при котором отсутствует возможность наблюдать индивидуальные шаги. Сохранение актуальных результатов может быть выполнено с помощью добавления ряда мест, в которых собирается информация о промежуточных результатах моделирования и которыми можно управлять, не влияя на процесс моделирования, например, добавляя одну или две дополнительные позиции для каждого из пяти переходов модели.

Второй способ записи результатов моделирования состоит в использовании программных вставок для записи отобранных результатов в выходной файл. Позже файл может читаться человеком или другой компьютерной программой. Чтобы увидеть использование выходного файла (и файла входа) в самой программе CPN Tools, необходимо сначала добавить несколько дополнительных деклараций.

Предопределенные функции INTxDATA.output и INTxDATA.input записывают и читают значение указанного выражения в указанном файле. При этом выражение должно иметь тип INTxDATA. Если выражение — мультимножество от INTxDATA, то нужно определять соответственно функции INTxDATA.output_ms и input_ms. При этом должно быть отмечено, что переменная, обозначающая пакеты, есть глобальная переменная и ее тип — массив INTxDATA. Это означает, что она будет связана с мультимножеством INTxDATA вместо единственного значения INTxDATA.

2.3.3. Моделирование сетей Петри в MATLAB

В системе MATLAB пока отсутствует специализированное универсальное приложение для работы с сетями Петри. В Интернете можно найти ряд пакетов программ, содержащих локальные версии комплекта инструментов для сетей Петри, например, разработанный в Чешском техническом университете графический редактор PN Editor [24]. Этот редактор позволяет рисовать сеть Петри и экспортировать ее в среду Petri Net Markup Language (PNML). Разработанное для сетей Петри приложение PN MATLAB Toolbox 2.0 поддерживает анализ поведения дискретных сетей Петри, непрерывных сетей Петри (CPN) и гибридных сетей Петри (HPN).

В работе [24] предлагается обычный подход исследования, основанный на трех шагах:

- модель в виде сети Петри изображается в графическом редакторе PN Editor;
- графический образ сети Петри экспортируется из редактора в матрицы формата MATLAB;
- с использованием системы MATLAB проводится анализ сети Петри и визуализация результатов моделирования (в MATLAB или в PN Editor).

Этот подход помогает организовывать работу модульным способом, использовать стандартные библиотеки и строить собственные инструменты. Другими словами, больше нет нужды в использовании сложного универсального инструмента, пользователь программирует собственный инструмент с поддержкой на этапах моделирования и визуализации. Это весьма удобно, потому что никакой инструмент при исследовании сложных систем не является универсальным.

Основные характеристики PN Editor:

- возможность рисовать разнообразные типы сетей Петри;
- независимая платформа;
- поддержка среды PNML;
- возможность расширения редактора дополнительными функциями.

PN Editor (рис. 2.13) был создан как легкий в использовании графический интерфейс в составе комплекта инструментов для сетей Петри. Он позволяет рисовать дискретные, непрерывные, гибридные сети Петри, а также расширенные гибридные сети Петри. PN Editor разработан на языке ЯВА.

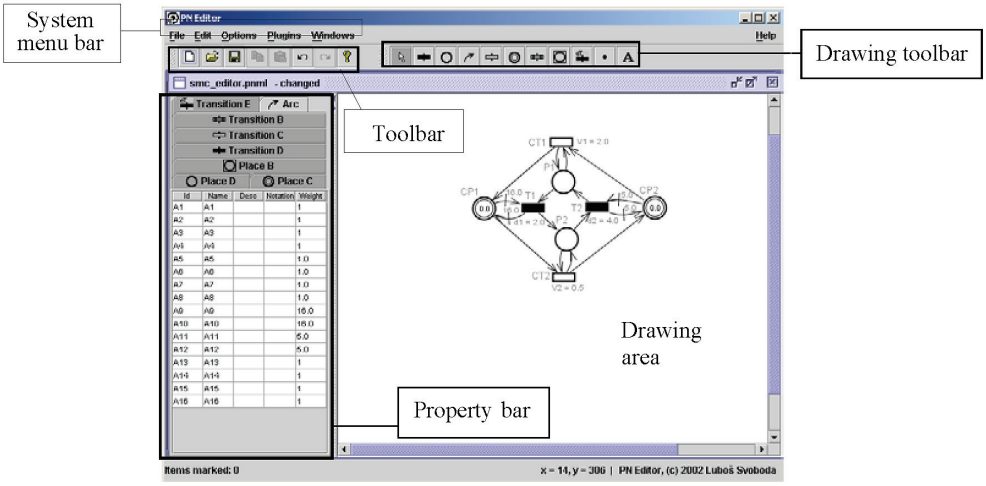


Рис. 2.13. Графический редактор PN Editor

PN Editor поддерживает язык PNML (базируется на XML-формате) и за счет импорта/экспорта файлов позволяет обеспечить максимальное сохранение совместимости с другими инструментами, поддерживающими PNML.

К сожалению, стандарт PNML полностью не определен для некоторых особых свойств сетей Петри или их внутренней информации. Этот недостаток преодолевается с помощью вставки элемента **toolspecific**, который определяет начало секции дополнительной информации, предназначенной только для данного редактора. Состав элемента **toolspecific** выражает, в каком редакторе дополнительная информация была разработана. Элементы, которые редактор не опознает, игнорируются. Значения по умолчанию установлены для элементов, которые отсутствуют в файле импорта.

Редактор первоначально был предназначен как среда для графической интерпретации сетей Петри, которая может быть преобразована к матричной форме, возможной для работы MATLAB. Хотя использование в PN Editor некоторых функций MATLAB ограничено, например показ всех его возможностей, редактор допускает пользовательские расширения за счет plugins. Например, он содержит plugins для показа графа достижимых маркировок и графа развития для непрерывной или гибридной сети Петри.

PN MATLAB Toolbox содержит функции для анализа свойств сети Петри. Есть функции для импорта сетей (матрицы Post, Pre, M_0), вычисление матрицы инциденций W, минимальных стандартизированных *P*-инвариантов и обнаружения тупиков. Комплект инструментов также содержит символический проигрыватель для временных и стохастических сетей Петри и функции для генерации на графе достижимых маркировок.

Кратко определим некоторые базовые функции PN MATLAB Toolbox. Следующие функции импортируют структуру и параметры сетей Петри, созданных в графическом редакторе. Функция **pnml2stpn.m** используется для импорта стохастических и временных сетей Петри, функция **pnml.m** используется в случае автономных сетей Петри, функция **pnml2hpn.m** — для импорта непрерывных и гибридных сетей Петри.

```
[Pre, Post, M0]=pnml (filename)
[Pre, Post, M0, d, TypeT]=pnml2STPN (filename)
[Pre, Post, M0, V, d, TypeP, nofpd] = pnml2HPN (filename)
```

Входной параметр*:*

filename — имя файла, в котором записана сеть Петри для ввода в PN Editor.

Выходные параметры:

Pre — матрица предусловий;
Post — матрица постусловий;
M0 — вектор начальной маркировки;
d — вектор задержек переходов;
V — вектор максимальных скоростей;
nofpd — число дискретных позиций;
TypeT — вектор типов переходов:
0 — переход с нулевым временем;
1 — переход с задержкой;
2 — переход со случайной задержкой (равномерное распределение);
TypeP — вектор типов позиций:
1 — дискретные позиции;
2 — непрерывные позиции.

Функция вывода **graph2hpn.m** используется, чтобы экспортировать результаты, рассчитанные в MATLAB, для его дальнейшей визуализации в графическом редакторе. Функция не имеет выходных параметров.

```
graph2hpn (Mar, VV, event_sum, time_event, mtimings, loopback,
nofpd, filename)
```

Входной параметр:

filename — имя файла, в котором записаны измененные параметры входной сети Петри в графическом редакторе;

Mar — матрица маркировок, каждый столбец матрицы состоит из дискретной и непрерывной частей. Дискретная часть представляет маркировку дискретных мест состояния, непрерывная часть представляет маркировку непрерывных мест в начале каждого состояния;

VV — матрица мгновенных скоростей переходов;

Time_XC — вектор относительных времен (*j*-й вход **Time_XC** соответствует *j*-му состоянию);

event_sum — вектор событий, каждый элемент которого — список событий, которые вызывали переход от одного состояния до следующего;

time_event — вектор времен появления специфических событий (время нахождения сети Петри в предыдущем состоянии);

mtimings — матрица моментов времени, каждый элемент столбца которой соответствует расчету временного вектора, запускающего задержку дискретного перехода (**nofpd** представляет число дискретных переходов);

loopback — индекс состояния после обхода петли (граф имеет предельный узел, когда q = **loopback**).

Функция **evolgr** генерирует динамику графа для непрерывных сетей Петри:

```
[Mar, VV, Time_XC] = evolgr (Pre, Post, M0, V, Prio)
```

Входные и выходные параметры были описаны выше.

Функция **HPN** генерирует динамику графа гибридной сети Петри. Результаты функции могут визуализироваться в Edit PN. Алгоритм функции был разработан для анализа поведения систем, смоделированных дискретными, непрерывными и гибридными сетями Петри, и может использоваться только для анализа сети с ограниченной дискретной частью. Алгоритм работает с единственной семантикой сервера, что предотвращает конфликты. Дискретные переходы не резервируют символы.

```
[Mar, VV, event_sum, time_event, mtimings, loopback] =
HPN (Pre, Post, M0, V, TypeP, d, Prio)
```

Входные и выходные параметры этой функции также были описаны выше.

Функция **drawnpl.m** показывает движение маркировки в процессе изменения всех состояний переходов графа. Эта функция обычно используется вместе с функцией, генерирующей динамику графа сети Петри, но может также использоваться отдельно.

```
drawnpl (Mar, time_event, loopback, nofpd, drawnDP, drawnCP, sizeF)
```

Входноые параметры*:*

drawnDP — вектор дискретных позиций для показа маркировки;

drawnCP — вектор непрерывных позиций для показа маркировки;

sizeF — размер шрифта текста в графе.

Пример 2.9. Преобразование гибридной сети Петри в PN MATLAB Toolbox [24].

В примере показывается последовательность шагов в преобразовании графа гибридной сети Петри, изображенной на рис. 2.14.

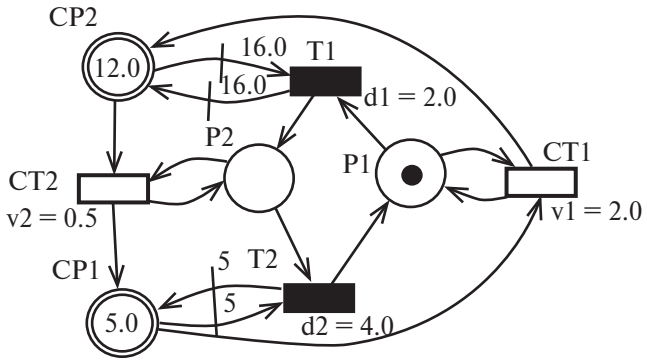


Рис. 2.14. Пример гибридной сети Петри

1. Представим сеть на рис. 2.14 в графической среде PN Editor.

2. Перенесем графическую информациию из редактора PN Editor в файле net.pnml.

3. Импортируем в среду MATLAB структуру сети с помощью функции `[Pre, Post, M0, C] = pnml2hpn (net.pnml)`.

4. Определим матрицы, необходимые для эволюции графа, преставленного на рис. 2.15.

```
[Mar, VV, event_sum, time_event, mtimings, loopback] =
HPN (Pre, Post, M0, V, TypeP, d, Prio).
```

а

$$Mar = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 5 & 1 & 0 & 0 & 5 & 7 \\ 12 & 16 & 17 & 17 & 12 & 10 \end{pmatrix}$$

$$VV = \begin{pmatrix} 2 & 2 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 \end{pmatrix}$$

$$event\_sum = \begin{pmatrix} [\,] \\ m(\mathrm{P4}) = 16 \\ m(CPI) = 0 \\ \mathrm{T1} \\ m(\mathrm{P3}) = 5 \\ \mathrm{T2} \\ m(\mathrm{P4}) = 16 \end{pmatrix}$$

$$mTimjngs = \begin{pmatrix} 2 & 2 & 1.5 & 2 & 2 & 2 & 2 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

$$Time\_event = (0 \quad 2 \quad 0.5 \quad 1.5 \quad 10 \quad 4 \quad 3)$$

$$loopback = 2$$

б

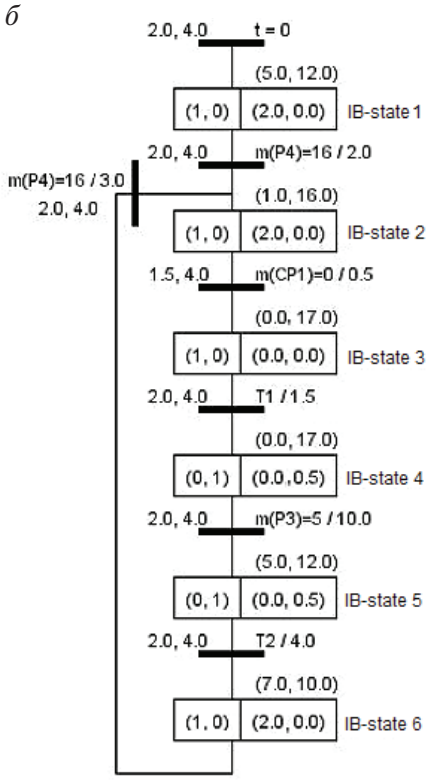


Рис. 2.15. Матрицы, создаваемые функцией `HPN` — *а*, эволюция графа, составленная согласно `HPN` — *б*

5. Проведем визуализацию динамики состояний графа в PN Editor. Выполним запуск PN Editor → в меню выбор пункта Plugins → выбор подпункта Evolution graph for HPN → выбор файла net.hpn.

```
Graph2hpn (Mar, VV, event_sum, time_event, mtimings, loopback,
nofpd, net.hpn)
```

На рис. 2.15, *б* представлена эволюция графа гибридной сети Петри (см. рис. 2.14).

6. Эволюция маркировки дискретной позиции P1 и непрерывных позиций CP1 и CP2 показана на рис. 2.16.

```
drawnpl (Mar, time_event, loopback, nofpc, drawnDP, drawnCP, sizeF)
```

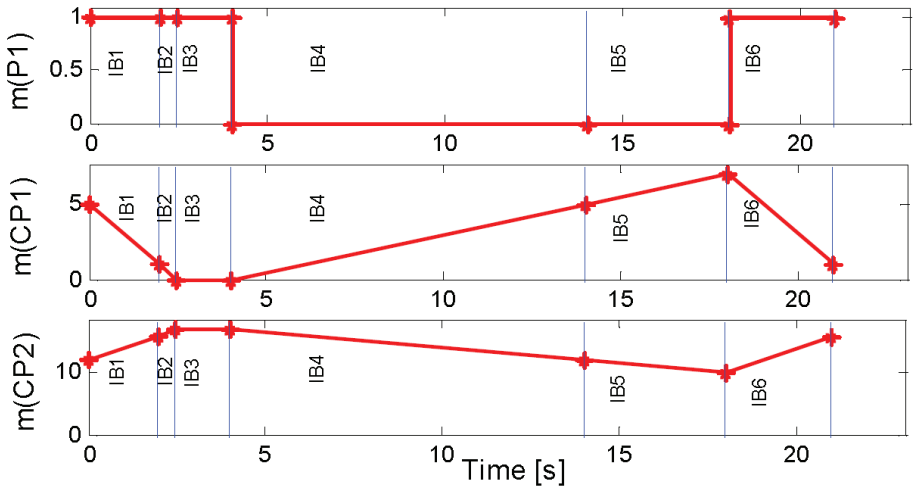


Рис. 2.16. Динамика маркировки избранных мест согласно **HPN** в зависимости от состояний IB

В заключении пункта представим еще один вариант локального (учебного) програмного инструмента для моделирования простых сетей Петри. Это графический интерфейс PNmodel (см. рис. 2.17), разработанный в среде системы моделирования MATLAB, который по таблице связей между позициями и переходами и начальной разметке M_0 показывает динамику работы сети.

Максимальное количество позиций и переходов ограничено 10, матрица связей имеет $10 \times 10 = 100$ элементов, которые служат на интерфейсе для ввода информации о свойствах соответствующих переходов. Реальное число позиций и переходов также указывается на интерфейсе. При двойном щелчке мышкой по клетке матрицы появляется дополнительное окно ввода числовых параметров свойств этого перехода. Например, на рис. 2.18 показано, каким образом вводятся данные о приоритете выбора той или иной позиции в переходе T1: выбирается приоритет выбора — **Неслучайный** или **Случайный** — и параметры и величины задержек времени.

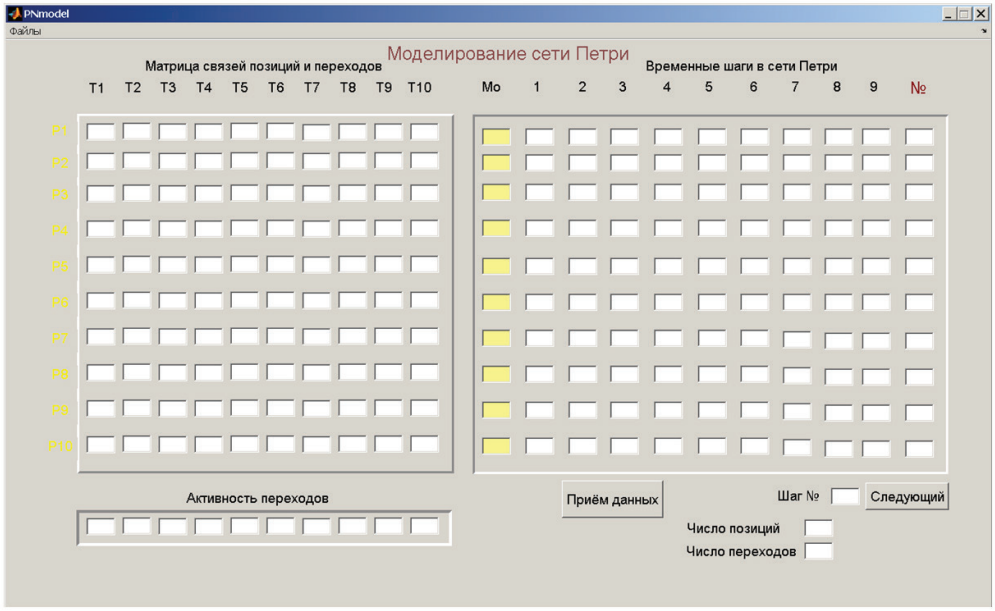


Рис. 2.17. Интерфейс учебного модуля «Моделирование простых сетей Петри»

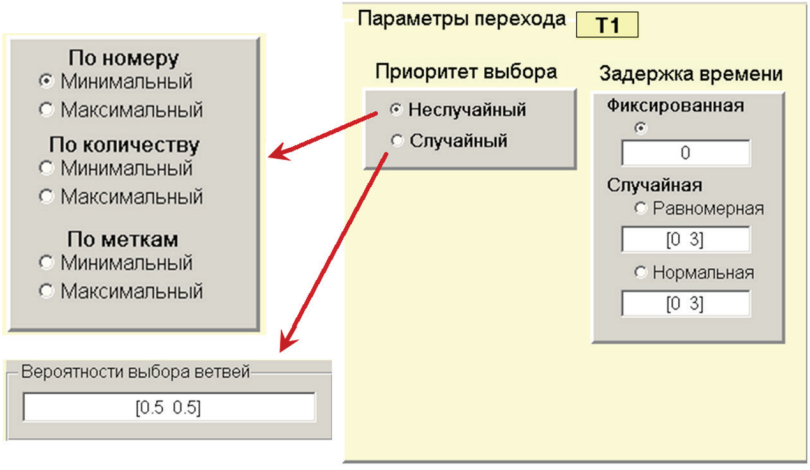


Рис. 2.18. Окно ввода параметров переходов

Порядок работы с описываемым интерфейсом достаточно прост. Сначала вводится число позиций и переходов, затем для каждого перехода в окне (рис. 2.18) устанавливаются правила выбора следующей позиции и время задержки, а для каждой позиции в главном окне (см. рис. 2.17) — начальная разметка M_0. Таким образом определяет-

ся сеть Петри. После нажатия клавиши **Прием данных** можно проводить ее моделирование, которое выполняется пошагово нажатием клавиши **Следующий**. При этом на первых 9 шагах в колонках интерфейса последовательно отображается число меток во всех позициях сети, а на следующих шагах — только в крайней правой колонке. Активность всех переходов на каждом шаге также показывается в элементах горизонтальной строки интерфейса.

Данные о результатах моделирования сети Петри отображаются в рабочем пространстве системы MATLAB и сохраняются в текстовом файле.

В заключение главы отметим нерешенные проблемы сетей Петри:

1. Проблема подмножеств. Даны две помеченных сети Петри. Является ли неразрешимой задача достижимости одной сети при достижимости другой сети как ее подмножества.

2. Сложность. Решение проблемы достижимости имеет экспоненциально возрастающие затраты по времени и такие же по числу состояний.

Вопросы и задания к главе 2

1. Назовите основные достоинства сетей Петри перед другими описаниями связей между элементами сложной системы.
2. Каковы составляющие графа сети Петри? Назовите их свойства.
3. Почему граф сети Петри называется мультиграфом?
4. Каким образом от графового представления сети Петри перейти к ее аналитическому описанию в виде массивов и функций?
5. Как с помощью сети Петри смоделировать причинно-следственные связи в некоторой сложной системе?
6. Как превратить статичную сеть Петри в динамичную?
7. Что такое маркировка сети Петри? Какова роль начальной маркировки?
8. Что такое правило срабатывания сети? Приведите примеры основных правил срабатывания.
9. Какие основные проблемы должны быть решены при анализе сетей Петри? Какие из них связаны с маркировкой сети, а какие — с переходами и состояниями?

10. Что означает проблема достижимости сетей Петри и каким образом она решается?
11. Какую проблему сетей Петри решает оценка живости переходов сетей Петри?
12. Каким образом безопасность сети Петри связана с динамикой маркировки сети?
13. Сформулируйте правила иерархического построения сетей Петри как относительно позиций, так и относительно переходов.
14. Что такое сателлитные сети Петри и чем они отличаются от иерархических сетей Петри?
15. Каким образом синхронизируются различные уровни сателлитных сетей Петри?
16. Дайте определение цветных сетей Петри. Какой тип меток должны иметь цветные сети Петри?
17. Какой недостаток примитивных сетей Петри устраняет введение временного механизма функционирования сети?
18. Поясните содержание метки $x : 2'(1,'\text{blue}') @ [12]$ некоторого мультимножества меток.
19. Определите функциональное содержание перехода, показанного на рис. 2.19.

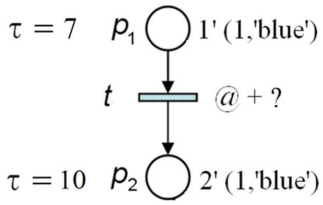


Рис. 2.19. Фрагмент цветной сети Петри

20. Дайте определение вложенных сетей Петри. Каким образом выполняется синхронизация в этих сетях?
21. Покажите, как будет меняться на интерфейсе PNmodel (см. рис. 2.17) динамика числа меток (состояний) позиций и активность переходов простейших сетей Петри, графы которых изображены на рис. 2.3 и рис. 2.4.

3. Модели систем на основе *E*-сетей

Построение математической модели для сколько-нибудь сложной системы почти всегда связано с нестандартными подходами при выборе методов и инструментов. Даже тогда, когда исследователь выбрал правильный инструмент, с его помощью бывает невозможно целиком охватить сложные переплетения событий и явлений процесса функционирования моделируемой системы. В этом случае модель становится такой же сложной, как сама моделируемая система [11, 25].

Естественное разрешение подобного противоречия — это декомпозиция системы на достаточно простые подсистемы или элементы и их изучение с учетом взаимодействия между ними и внешней средой. При этом возможны варианты как классического, так и системного подходов, описанные ранее в [11]. Очевидно, чтобы задать сложную систему, необходимо представить описание всех ее элементов и взаимодействия между элементами. Наиболее часто используемой формой описания является объектно-ориентированное представление элементов, получающих сообщения и изменяющих свои внутренние состояния в зависимости от полученного сообщения. Модельное описание какой-либо подсистемы определяется в данном случае как объект или логическое единство составляющих его частей и функциональных связей между ними. Внутреннее устройство и внешние связи объекта определяются содержанием выполняемых операций обработки в соответствии с принятым уровнем детализации анализа данной части предметной области. Предполагается, что объект обладает внутренней памятью и набором операций, причем над внутренней памятью могут выполняться операции только этого набора. Обобщенное графическое представление такого объекта показано на рис. 3.1.

Объект может реагировать на определенную совокупность сообщений **X**, каждое из которых выражает запрос на выполнение указанных в нем операций. При этом указываются вид операций и необходимые

для выполнения аргументы, а каким образом они должны выполняться — в сообщении не отражается. Характер выполнения операций определяется объектом, получающим сообщение. Сообщение является единственным средством инициирования операций над объектом. Таким образом, принимающий данное сообщение объект должен содержать механизм, который будет опознавать сообщение, выбирать соответствующие ему операции и передавать содержащиеся в сообщении аргументы для выполнения.

Рис. 3.1. Объект *E*-сети

Набор выполняемых операций $\mathbf{Y} = \boldsymbol{F}(\mathbf{X})$ объекта можно менять в зависимости от условий применения объектов. Состояние объекта (значения переменных конкретного экземпляра) запоминается после того, как объект перестает быть активным, и может быть использовано в дальнейшем.

Итак, объект — это логически законченный модуль, изменяющий свое внутреннее состояние, отражаемое вектором выходных параметров **Y** и зависящее от вектора входных воздействий **X**. Для достижения целей математического моделирования сложные системы требуется представить в виде логико-математического описания взаимодействующих объектов в форме «вход — состояние — выход». В качестве аппарата математического моделирования предлагается использовать аппарат *E*-сетей как наиболее подходящий для описания семантики строения и взаимодействия описанных выше объектов.

Элементарные функции можно описать в виде математических преобразований, которые легко описываются аппаратом *E*-сетевого моделирования, где параметрическая часть, отражающая хранение и передачу информации, представляется позициями (на рис. 3.2 это позиция *E*), а математическая, отражающая преобразование параметрической части, представляется переходами (на рис. 3.2 это переход *P*) [26, 27].

Поэтому название представленной модели в виде *E*-сети отображает главное ее свойство — связь через вычисления (англ. *evaluation* —

вычисление, оценка) между позициями и переходами на графе (между входом и выходом структурной схемы). Другое название моделей в виде *E*-сетей — *вычисляемые* сети или *оценочные* сети.

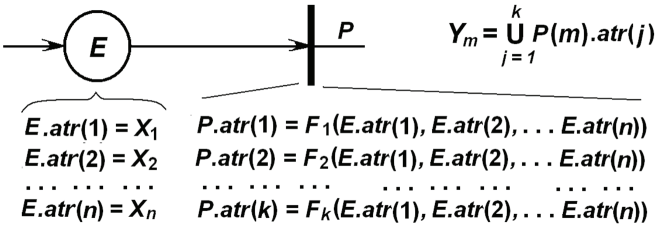


Рис. 3.2. Элементы графа *E*-сети

3.1. Формализм *E*-сетей

E-сети как результат развития аппарата примитивных сетей Петри представляют собой одно из наиболее мощных расширений этого подхода [26, 27]. В отличие от примитивных сетей Петри в *E*-сетях:

- имеются несколько типов вершин-позиций: простые позиции, позиции-очереди, разрешающие позиции;
- метки (фишки) могут снабжаться набором признаков (атрибутов, свойств);
- с каждым переходом может быть связана ненулевая задержка и функция преобразования атрибутов меток;
- введены дополнительные виды вершин-переходов;
- в любую позицию может входить не более одной ветви и выходить также не более одной.

Формально *E*-сеть задается пятеркой вида

$$E = < P, P_p, P_r, T, M_0 >, \qquad (3.1)$$

где P — множество безусловных позиций; P_r — множество решающих позиций; P_p — множество макропозиций; T — множество переходов $\{t_i\}$, причем $t_i = (S_i, \rho_i, \tau_i)$, где S_i — тип одного из пяти вариантов переходов (S_i э {1...5}); $\rho_i = \rho\ (t_i)$ — функция преобразования атрибутов меток; $\tau_i = \tau\ (t_i)$ — функция, определяющая длительность срабатывания перехода.

Существует несколько типов позиций [26]:

- *простые* позиции, могут содержать не более одной метки и изображаются кружком;
- *разрешающие* позиции, выполняют управляющую функцию, определяющую направление перемещения меток, их изображения помимо кружков содержат квадраты, обозначающие логическую процедуру управления;
- *макропозиции*, которые могут содержать произвольное число меток и изображаются специальными графическими объектами (овалами, кругами и др.).

Переходы в *E*-сетях изображаются отрезком прямой линии и могут быть ассоциированы со следующими процедурами:

- *процедура вычисления временной задержки* $\tau(t_i)$;
- *процедура преобразования атрибутов* меток $\rho(t_i)$, проходящих через переход;
- *разрешающая процедура*, если переход имеет входную разрешающую позицию.

Следует отметить, что для усиления выразительных средств *E*-сетей разрешающие процедуры, процедуры преобразования атрибутов, сетевых переменных и процедуры вычисления временных задержек могут иметь абсолютно произвольный вид и реализовывать любые вспомогательные вычисления и действия, необходимые пользователю.

Еще одно важное отличие *E*-сетей от сетей Петри состоит в том, что метки интерпретируются как *транзакты*, перемещающиеся по сети, а переходы трактуются как устройства, выполняющие ту или иную обработку транзактов. Поэтому ни одна позиция *E*-сети не может содержать более одной метки, т. е. любая *E*-сеть изначально является безопасной.

Переходы *E*-сети описаны ниже в виде функциональных преобразований и соответствующих графических форматов. В первую очередь это три безусловных перехода:

T-переход или простой переход («исполнение»). Его графическое представление аналогично представлению вершины-перехода сети Петри (см. рис. 3.3, *а*). Переход *T* работает при наличии метки во входной позиции и отсутствии ее в выходной. Формально это можно записать так: $(1; 0) \mid\!\rightarrow (0; 1)$. Этот переход позволяет отразить в модели занятость некоторого устройства (подсистемы) в течение некоторого времени, определяемого параметром $\tau(t_j)$;

F-переход («разветвление»). Графическое представление приведено на рис. 3.3, *б*. Срабатывает *F*-переход при тех же условиях, что и *T*-переход: $(1;0,0)\mid\!\rightarrow(0;1,1)$. С содержательной точки зрения *F*-переход отображает разветвление потока информации (транзактов) в системе;

J-переход — конъюнкция («объединение»). Графическое обозначение показано на рис. 3.3, *в*, переход срабатывает при наличии меток в обеих входных позициях и отсутствии в выходной позиции: $(1,1;0)\mid\!\rightarrow(0,0;1)$.

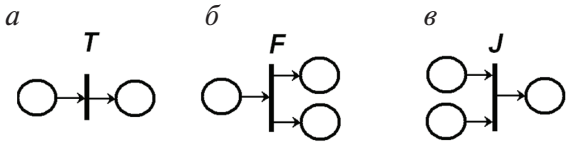


Рис. 3.3. Безусловные переходы *E*-сети

В *E*-сети имеются два перехода с разрешительными процедурами, имеющими булевские состояния («истина» или «ложь»):

X-переход образует объединение потоков при наличии нескольких условий, определяющих некоторое событие («переключатель»). По сравнению с тремя предыдущими типами переходов он содержит дополнительную управляющую («разрешающую») позицию, которая графически обозначается обычно либо квадратиком, либо шестиугольником слева от его изображения (рис. 3.4, *а*).

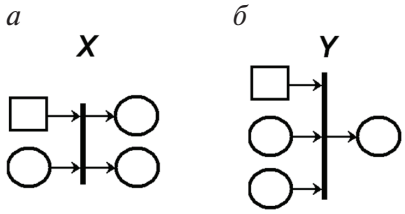


Рис. 3.4. Переходы *E*-сети с разрешительными процедурами

Логика его срабатывания задается следующими соотношениями: при наличии метки во входной позиции она перемещается на верхнюю выходную позицию при нулевом разрешающем уровне и на нижнюю выходную позицию — при единичном разрешающем уровне (в описании процедуры состояние разрешительного условия выделено полужирным шрифтом):

$$(\mathbf{0};1;\ 0,0)\,|\!\rightarrow(\mathbf{0};0;\ 1,0),$$
$$(\mathbf{0};1;\ 0,1)\,|\!\rightarrow(\mathbf{0};0;\ 1,1),$$
$$(\mathbf{1};1;\ 0,0)\,|\!\rightarrow(\mathbf{0};0;\ 0,1),$$
$$(\mathbf{1};1;\ 1,0)\,|\!\rightarrow(\mathbf{0};0;\ 1,1).$$

X-переход изменяет направление потока информации (транзактов). В общем случае разрешающая процедура может быть сколь угодно сложной, но сущность ее работы заключается в проверке выполнения условий разветвления потока (с точки зрения программиста, разрешающая позиция аналогична условной инструкции типа `if`);

Y-переход («выбор», «приоритетный выбор»). Этот переход также содержит разрешающую позицию (см. рис. 3.4, *б*). Логика срабатывания этого *Y*-перехода состоит в следующем: переход срабатывает в том случае, когда во входных позициях имеется хотя бы одна метка. При отсутствии разрешения (состояние «ложь») метка в выходную позицию передается от самой старшей (верхней на рис. 3.4, *б*) помеченной позиции, при наличии разрешения (состояние «истина») — от самой младшей (нижней на рис. 3.4, *б*) помеченной позиции:

$$(\mathbf{0};\ 1,\ 1;\ 0)\,|\!\rightarrow(\mathbf{0};\ 0,1;\ 1)$$
$$(\mathbf{0};\ 1,0;\ 0)\,|\!\rightarrow(\mathbf{0};\ 0,0;\ 1)$$
$$(\mathbf{0};\ 0,1;\ 0)\,|\!\rightarrow(\mathbf{0};\ 0,0;\ 1)$$
$$(\mathbf{1};\ 1,\ 1;\ 0)\,|\!\rightarrow(\mathbf{0};\ 1,0;\ 1)$$
$$(\mathbf{1};\ 1,0;\ 0)\,|\!\rightarrow(\mathbf{0};\ 0,0;\ 1)$$
$$(\mathbf{1};\ 0,1;\ 0)\,|\!\rightarrow(\mathbf{0};\ 0,0;\ 1).$$

Неиспользованные метки во входных позициях сохраняются.

Y-переход отражает приоритетность одних потоков информации (транзактов) по сравнению с другими. При этом разрешающая процедура может быть определена различным образом: как операция сравнения фиксированных приоритетов меток; как функция от атрибутов меток (например, чем меньше время обслуживания, тем выше приоритет). В некотором смысле *Y*-переход работает аналогично инструкции выбора типа `case`.

Поскольку в *E*-сети все переходы обладают свойством безопасности, то это означает, что в выходных позициях (которые, в свою очередь, могут быть входными для следующего перехода) никогда не может быть более одной метки. Более того, чтобы любой из описанных

выше пяти переходов сработал, необходимым условием является отсутствие меток в выходных позициях. Вместе с тем, в *E*-сетях существуют понятия *макроперехода* и *макропозиции,* которые позволяют отображать в модели процессы накопления обслуживаемых транзактов в тех или иных узлах системы, а также расширить логические возможности *E*-сетей.

Рассмотрим некоторые из них.

Макропозиция *Очередь* представляет собой линейную композицию *T*-переходов, и суммарное количество выходных вершин-позиций определяет «емкость» очереди (рис. 3.5).

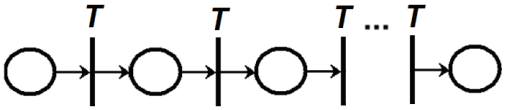


Рис. 3.5. Макропозиция *Очередь*

Аналогичным образом, путем композиции *N* однотипных переходов могут быть получены макропереходы всех типов: *JN, XN*, *YN*.

Макропозиция *Генератор* (рис. 3.6, *а*) позволяет представлять в сети источник меток (транзактов). Если необходимо задать закон формирования меток, то генератор может быть дополнен разрешающей позицией (рис. 3.6, *б*).

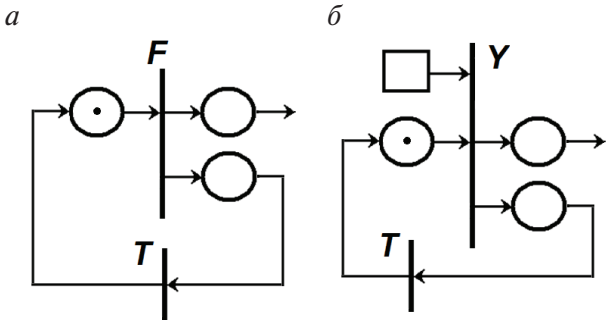


Рис. 3.6. Макропозиция *Генератор*

Макропозиция *Аккумулятор*. Поскольку в *E*-сети нельзя «накапливать» метки, то вводится макропозиция поглощения или *Аккумулятор* (см. рис. 3.7).

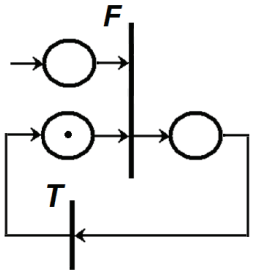


Рис. 3.7. Макропозиция *Аккумулятор*

В целях повышения компактности и наглядности *E*-сети для обозначения макропозиций используют специальные символы и графические образы, представленные в табл. 3.1.

Таблица 3.1

Макропозиции *E*-сетей

Название макропозиции	Символьное обозначение	Графический образ
Очередь	*Q*	*Q*
Генератор	*G*	*G*
Аккумулятор	*A*	*A*

Пример 3.1. Модель мультипрограммной системы в виде *E*-сети.

Ниже на рис. 3.8 приведена модель мультипрограммной вычислительной системы, построенная в виде *E*-сети. Обработка поступающих заданий организована в ней по принципу квантования времени: каждому заданию выделяется равный отрезок (квант) процессорного времени. Если задание выполнено, то оно покидает систему, если же времени оказалось недостаточно, то задание встает в очередь и ждет повторного выделения кванта времени.

На рисунке переходы T_j, соответствующие определенным событиям в системе, имеют следующие обозначения: T_1 — постановка задания в очередь; T_2 — выполнение задания в течение одного кванта времени; T_3 — анализ степени завершенности задания. Разрешающие позиции R_1 и R_2 служат для задания закона формирования случайных интервалов времени между поступающими заданиями и интервалов времени, необходимых для полного обслуживания каждого из них. Очевидно,

что основная часть «интеллекта» этой модели сосредоточена в правилах управления разрешающими позициями R_1 и R_2, которые должны быть представлены в виде текстов на некотором встроенном языке выбранного инструмента моделирования *E*-сетей.

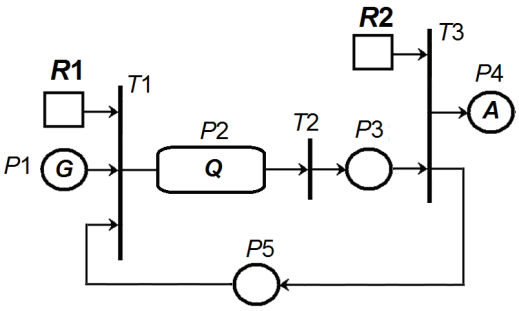


Рис. 3.8. Модель мультипрограммной вычислительной системы

Следовательно, в терминах *E*-сети сложную систему можно представить в виде программного объекта, который будет описываться следующими свойствами:

- вектор входных воздействий;
- набор логико-математических операций для каждого перехода *E*-сети;
- вектор атрибутов метки для каждой маркированной позиции *E*-сети;
- вектор выходных параметров.

Кроме того, любая модель в терминах *E*-сети должна иметь начальную маркировку, которая отражает начальное состояние модели в момент запуска. Считая *E*-сетевую модель программно-логическим отображением физического объекта, можно определить загрузку текущего состояния объекта как набор атрибутов меток начальной маркировки *E*-сети, а для сохранения состояния объекта использовать обновленную информацию в позициях начальной маркировки.

Для этой цели требуется организовать граф и математическое описание преобразований в переходах *E*-сетевой модели таким образом, чтобы маркировка имитационной модели в процессе ее функционирования всегда возвращалась к начальной через N тактов моделирования. Значение N определяется как цикл работы *E*-сетевой модели. Под циклом работы *E*-сетевой модели будем понимать количество модельных тактов, требуемых для возврата к начальной маркировке

в процессе последовательного изменения маркировки *E*-сети, определяемого логикой срабатывания разрешенных переходов. Цикл работы модели может быть различен в различные моменты времени, но в соответствии с логикой функционирования модели, представленной на рис. 3.9, можно сделать вывод, что выходная метка модели будет определять окончание цикла работы модели. Следовательно, считывание состояния модели можно организовать при появлении метки на выходе *E*-сети, описывающей моделируемый объект.

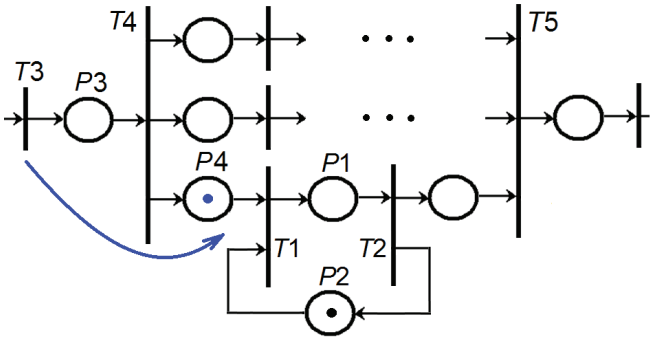


Рис. 3.9. Использование атрибутов метки для хранения информации

Информация в *E*-сети распространяется посредством атрибутов метки, движущейся по *E*-сетевому графу в процессе последовательного изменения маркировки *E*-сети. Таким образом, информация в *E*-сети хранится в виде атрибутов меток. Для считывания и записи состояния модели требуется организовать *E*-сетевой граф так, чтобы маркировки до и после цикла работы модели были идентичны, а также позиции, определяющие состояние модели, были промаркированы. Данная задача решается введением обратных связей на участках, определяющих состояние модели (рис. 3.9).

3.2. Методика применения *E*-сетей в моделировании систем

В основе *E*-сетевого моделирования лежит, фактически, событийно управляемое моделирование, что в совокупности с методами генерации случайных чисел для задания времени срабатывания переходов

представляет собой один из наиболее распространенных способов моделирования. Тот факт, что модель управляется событиями, свидетельствует о том, что состояние системы изменяется только при срабатывании переходов и остается неизменным между срабатываниями. Так как время срабатывания переходов часто является случайной величиной, то для получения статистически достоверных результатов требуется соответствующее количество запусков модели.

Пример 3.2. Хранение и передача информации в *E*-сети.

На рис. 3.9 изображен фрагмент графа некоторой *E*-сети. На нем позиция *P*2 является маркированной, а атрибуты метки в данной позиции определяют, допустим, заголовок пакета данных. В процессе функционирования *E*-сети возможно срабатывание перехода *T*1 на временном шаге *i*. Для этого на шаге (*i*-1) должна появиться метка инициализации в позиции *P*3 и через переход *T*4 перейти в позицию *P*4 (маршрут метки инициализации отмечен на графе стрелкой). Срабатывание перехода *T*1 типа *J* обязательно приведет к срабатыванию перехода *T*2, так как условием срабатывания перехода *T*2 является наличие метки в позиции *P*1, а постусловием срабатывания перехода *T*1 является появление метки в позиции *P*1. При этом маркированная позиция *P*2 станет немаркированной. Срабатывание перехода *T*2 приведет к появлению метки в позиции *P*2, так как постусловием срабатывания перехода *T*2 типа *F*, в соответствии с принципами работы *E*-сети, является появление меток во всех позициях, являющихся выходными для данного перехода. Появление метки в позиции *P*2 является возвратом к начальной маркировке и, тем самым, сохранением заголовка пакета данных на одном временном шаге. Информация о заголовке может быть передана на выход через переход *T*5 при условии его срабатывания.

Таким образом, аппарат *E*-сетевого моделирования предоставляет возможность сохранять и восстанавливать состояние объекта каждые *N* тактов моделирования, загружать вектор входных воздействий, а также формировать вектор выходных параметров объекта.

Среди преимуществ *E*-сетей одной из важнейших является возможность адекватного описания систем с параллельно функционирующими компонентами. Например, в работе [28] Томского политехнического университета представлены *E*-сетевые модели процессов отказов-восстановлений последовательно-параллельных технических

систем для различных типов параллельного резервирования и, кроме того, учтена возможность моделирования надежности восстанавливаемых объектов.

Пример 3.3. Построение и анализ сложной технической системы с резервированием на основе формализма *E*-сетей [28].

Для более полного представления о содержании задачи моделирования введем некоторые определения из теории надежности.

Надежность компонента — это вероятность того, что он будет выполнять свою функцию в течение определенного промежутка времени при работе в нормальных (или заданных) внешних условиях.

Отказ — событие или нерабочее состояние, при котором компонент или его часть не работает или не может работать как заранее определено.

Вероятность безотказной работы — вероятность того, что элемент способен выполнять свою функцию в течение определенного промежутка времени при заданных условиях; что при заданных условиях работы в интервале безотказной работы системы отказ не возникнет.

Говорят, что компоненты системы соединены *последовательно,* если для работы системы необходимо, чтобы каждый из них работал, т. е. отказ любого компонента вызывает отказ системы. Когда надежность последовательной системы не удовлетворяет проектным требованиям и улучшение составных компонентов невозможно (более надежные части не доступны или слишком дороги), становится необходимым действовать на структурном уровне и использовать резервированные конфигурации. Говорят, что конфигурация системы *резервированная* (*параллельная*), когда отказ компонента не обязательно ведет к отказу системы.

Существует три типа параллельного резервирования [28]:

1. При *горячем резервировании* все *M* компонентов модуля находятся в рабочем состоянии или *«полностью нагружены»* при использовании. Это означает, что все они стареют одновременно.

2. При *холодном резервировании* используется один компонент, а остальные (*M*-1) компонентов модуля находятся в резерве. Когда используемый компонент отказывает, один из резервных компонентов подключается с помощью механизма переключения.

3. При *недогруженном резервировании* резервные компоненты с течением времени стареют. Это эквивалентно ситуации, когда резерв

может рассматриваться как частично нагруженный в отличие от полностью нагруженного при горячем резервировании и не нагруженного при холодном.

Системы называются *восстанавливаемыми*, если основной и резервные компоненты могут восстанавливаться после отказа.

Теперь рассмотрим *E*-сеть для компонента параллельной системы с горячим резервированием без восстановления (рис. 3.10). В ней имеется два *X*-перехода — *R*1-*T*2→(*S*7-*S*5) и *R*3-*T*6→(*S*1-*S*6) — и один *Y*-переход *R*2-*T*5→*S*8. Состоянию горячего резервирования соответствует позиция *S*1 — компонент находится в резерве, состоянию *S*7 — отказ компонента. Только один переход *T*1 обладает случайной временной задержкой — временем до отказа из состояния резерва. Временная задержка остальных переходов равна нулю. В начале моделирования метка находится в позиции *S*1.

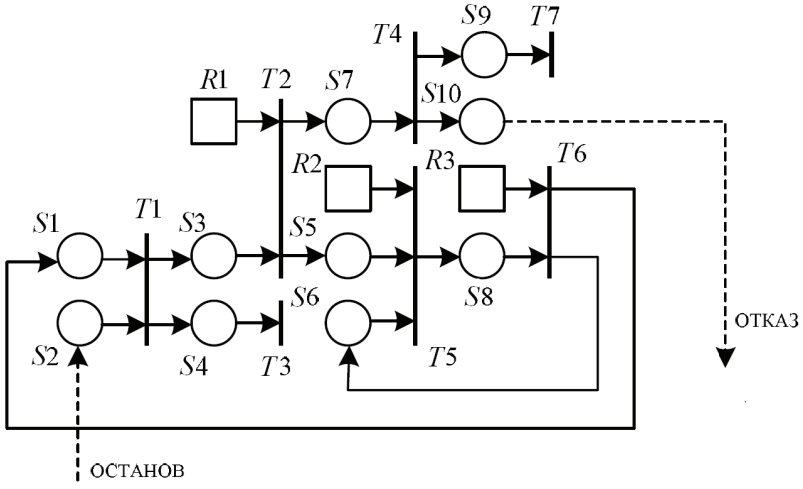


Рис. 3.10. *E*-сеть для компонента параллельной системы с горячим резервированием без восстановления

Рассмотрим возможные маршруты метки, предполагая, что первоначально она находится в позиции *S*1. Срабатывает переход *T*1 типа *J* и метка переходит на позиции *S*3 (основная) и *S*4 (дублирующая). По решающему правилу *R*1 по истечении случайного времени t_2 перехода *T*2 типа *X* компонент отказывает. Через переход *T*4 на управляющее устройство (УУ) посылается сигнал об отказе компонента (позиция *S*10). Если модельное время t_m меньше времени t_2, то мет-

ка переходит на позицию *S*5, далее срабатывает в соответствии с правилом *R*2 переход *T*5 типа *Y* и метка появляется в позиции *S*8. Здесь она через переходы *T*6 и T*5* зацикливается между позициями *S*8 и *S*6, а также транслируется в позицию *S*1 исходного горячего резерва. Цикл повторяется согласно дискретным шагам модельного времени, пока не сработает по правилу *R*1 переход *T*2 и перенаправит метку в позицию *S*7. Отметим, что при моделировании по событиям модельное время меняется только в момент наступления очередного ближайшего события (т. е. $t_m = t_2$) наличие переходов *T*5 и *T*6 с инцидентными позициями на графе *E*-сети не нужно. В этом случае остается решающий переход *T*2 и вспомогательные переходы *T*1 и *T*4, служащие для регистрации состояния сети.

В качестве расширения рассмотренной выше сети рассмотрим *E*-сеть, построенную для компонента резервированной группы с недогруженным резервированием и восстановлением (рис. 3.11) [28].

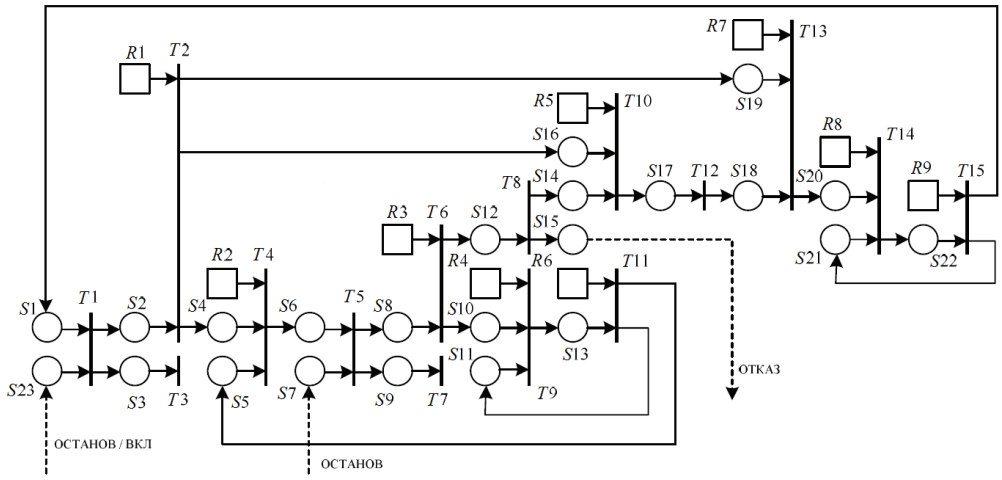


Рис. 3.11. *E*-сеть для компонента параллельной системы

Трем состояниям компонента соответствуют три позиции: *S*1 — компонент находится в резерве, *S*6 — компонент в рабочем состоянии, *S*17 — компонент отказал и восстанавливается. Только три перехода обладают случайной временной задержкой: *T*1 — время до отказа из состояния резерва, *T*5 — время до отказа из рабочего состояния, *T*12 — время до восстановления. Временная задержка остальных переходов равна нулю. В начале моделирования метка может находить-

ся в позиции *S*6, если компонент является основным, и в позиции *S*1, если он резервный.

Рассмотрим возможные маршруты метки, предполагая, что первоначально она находится в позиции *S*6 (основной компонент). По истечении случайного времени t_5 перехода *T*4 метка передается в позицию *S*6 и компонент отказывает. Метка проходит по маршруту *S*8–*S*12–*S*14–*S*17, компонент становится на восстановление, которое занимает время t_{12}, связанное с переходом *T*12.

При переходе через *T*8 на управляющее устройство (УУ) посылается сигнал об отказе компонента (позиция *S*15). После восстановления метка проходит маршрут *S*18–*S*20–*S*1 и компонент занимает место в резерве. Находясь в недогруженном резерве, компонент может отказать через случайное время $t_1 >> t_5$. Тогда метка проследует по маршруту *S*1–*S*2–*S*16–*S*17 и далее (после восстановления) *S*18–*S*20–*S*1, и компонент снова переходит в резерв.

Резервный компонент может также получить от УУ сигнал ВКЛ на подключение в рабочее состояние (метка в позицию *S*23). Это происходит, если остальные компоненты резервной группы отказали. Тогда нормальная работа перехода *T*1 прерывается и метка из *S*1 переходит по маршруту *S*1–*S*2–*S*4–*S*6 в рабочее состояние.

Возможна ситуация, когда работающему компоненту от УУ поступает сигнал ОСТАНОВ, прерывающий моделирование (метка в позицию *S*7). Это происходит, если система отказала по причине выхода из строя какого-нибудь последовательно соединенного компонента. Тогда в атрибут метки записывается оставшееся время моделирования, нормальная работа перехода *T*5 прерывается, и метка следует по маршруту *S*6–*S*8–*S*10 и далее зацикливается в позициях *S*13–*S*11 до тех пор, пока работа системы не восстановится. Тогда метка вновь занимает позицию *S*6 через *S*13–*S*5, и моделирование продолжается.

Аналогичная ситуация может произойти, когда компонент находится в резерве. Метка проследует по маршруту *S*2–*S*19–*S*20 и по циклу в позициях *S*22–*S*21. При восстановлении работоспособности системы метка из *S*22 поступает вновь в *S*1 (резерв). Различие между сигналами ВКЛ и ОСТАНОВ для резерва определяется по атрибуту метки в позиции *S*23.

Если система отказала во время восстановления компонента, то после его восстановления метка ожидает разрешения (цикл *S*22–*S*21) проследовать в позицию *S*1 (резерв).

Большой размер данной сети объясняется тем, что недогруженное резервирование с восстановлением является наиболее общим и сложным случаем. В случаях горячего и холодного резервирования, а также резервирования без восстановления, сети существенно упрощаются и получаются путем удаления лишних позиций, переходов и связанных с ними ветвей (см. рис. 3.10).

3.3. Программные инструменты моделирования *E*-сетей

Как в случае сетей Петри, универсального инструмента моделирования *E*-сетей не существует. Отсутствует специализированное приложение в системе технического моделирования MATLAB, автору неизвестны и другие системы моделирования с явно выраженной направленностью на моделирование *E*-сетей.

В то же время в высших учебных заведениях России уже много лет существуют научные школы, активно использующие методологию моделирования сложных систем с помощью *E*-сетей. В первую очередь это преподаватели и научные сотрудники Томского политехнического университета [27, 28, 29], Томского государственного университета систем управления и радиоэлектроники [30], также Донского государственного технического университета [31].

В этом параграфе рассмотрим наиболее интересный с методической точки зрения вариант разработки программной системы имитационного моделирования *E*-сетей под названием EVA (E-net Valuation Adviser) [29], разработанной в Томском политехническом университете в начале 90-х годов XX века. Хотя графические и программные возможности того времени были ограниченными, эта программа может служить хорошим прототипом для разработки новых средств моделирования *E*-сетей на основе современных компьютерных технологий.

Система моделирования EVA имеет в своем составе следующие программные модули:

1) графический редактор EEE для задания структуры *E*-сети, описывающей моделируемую систему;

2) препроцессор PREPARE языка EDEF, который, используя определения *E*-сети, созданные с помощью редакторов EEE и EDIT, генерирует описание моделируемой системы на языке Си;

3) модуль MODEL на языке Си, который в качестве своей составной части включает текст описания модели, подготовленный препроцессором PREPARE, и после компиляции представляет собой исполняемый файл программы интерпретации *E*-сети, описывающей работу моделируемой системы;

4) графический отладчик GRDEB, который в случае отладки модели в графическом режиме отображения работы *E*-сети используется вместо модуля MODEL;

5) компилятор TCC и редактор связей TLINK языка Turbo C;

6) другие вспомогательные компоненты, которые используются неявно, например пакетные файлы, с помощью которых вызываются на исполнение программы PREPARE, TCC и TLINK применительно к указанной модели, тем самым создавая исполняемый файл модели.

Описательная часть *E*-сети (текстовый файл с расширением DEF), включающая определение длительностей активных фаз переходов, описание процедур преобразования атрибутов, выбор разрешающих процедур и т. п., создается с помощью специального языка EDEF, разработанного на базе языка Си. В качестве текстового редактора необходимо использовать любой ASCII-редактор, имеющийся на персональной ЭВМ пользователя.

Работа по созданию модели заключается в задании двух исходных файлов описания модели и их последовательного преобразования до получения исполняемого файла модели системы. На рис. 3.12 приводится схема процесса создания модели и взаимосвязей между файлами (в предположении, что модель имеет имя NET).

Участок, выделенный штриховыми линиями, означает, что работа с указанными программами и файлами автоматизирована с помощью пакетного файла UNION. BAT. Двойной рамкой представлены программные компоненты системы моделирования EVA, а одинарной рамкой — файлы, описывающие моделируемую систему.

После задания структуры *E*-сети и сохранения информации о ней в файле (если модели дано имя NET, то файл имеет имя NET. EEE) с помощью графического редактора EEE можно перейти к определению описательной части сети. Для этого используется специальный язык EDEF. Ниже приводится не совсем формальное, но довольно строгое описание этого языка.

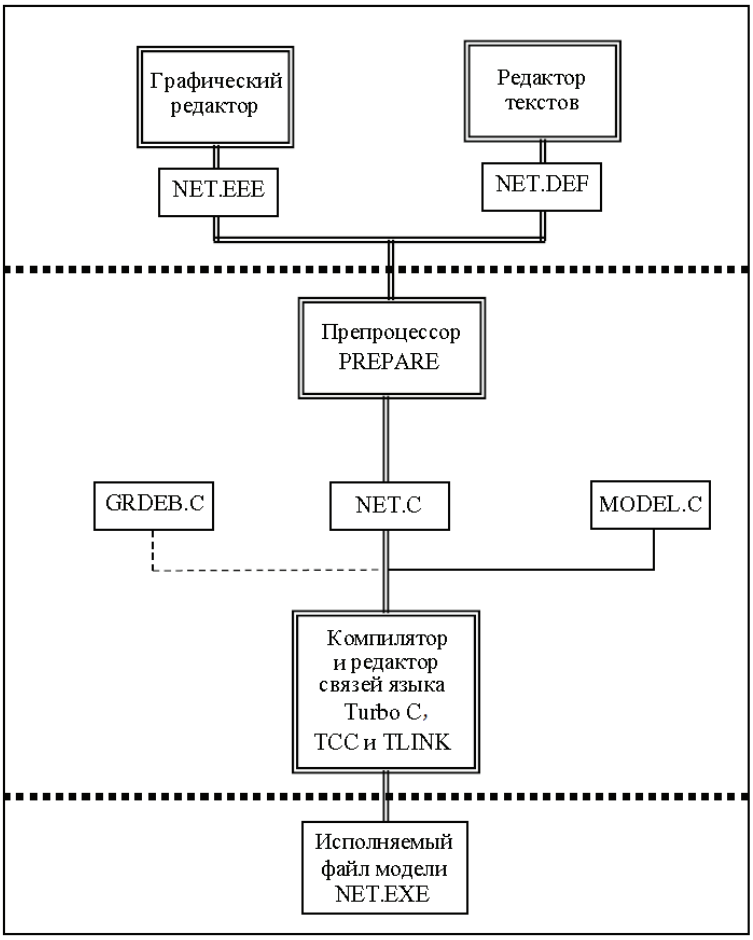


Рис. 3.12. Схема процесса создания модели и взаимосвязи между файлами

```
<программа> ::= [<комментарии>]<разделы>
<комментарий> ::= <произвольный текст без символов двоеточий «:»>
<разделы> ::= [<раздел A>][<раздел I>][<раздел R>][<раздел D>]
              [<раздел> T][<раздел S>][<раздел E>]
<раздел A> ::= A: <заголовки, объявления переменных и функций
             языка Си>
<раздел I> ::= I: <составной оператор языка Си>
<раздел R> ::= R: <номер перехода><составной оператор языка Си>
              [<номер перехода><составной оператор языка Си>]...
<раздел D> ::= D: <номер перехода><составной оператор языка Си>
              [<номер перехода><составной оператор языка Си>]...
```

```
<раздел T> ::= T: <номер перехода><составной оператор языка Си>
               [<номер перехода><составной оператор языка Си>]...
<номер перехода> ::= T<число>:
<раздел S> ::= S: <функция статистики>[<функция статистики>]...
<функция статистики> ::= <функция stat_s> | <функция stat_a> |
              <функция stat_q> | <функция stat_t>
<раздел E> ::= E: <составной оператор языка Си>
```

Программа на этом языке представляет собой набор разделов, каждый из которых является фрагментом программы на языке Си. Каждый раздел в программе на языке EDEF может встречаться не более одного раза и при отсутствии необходимости в нем может быть опущен. Переменные, используемые в них, можно разделить на два класса: пользовательские и системные. Пользовательские используются при необходимости, а системные определяют состояние и параметры имитационной модели. Основными системными переменными, которыми исследователь может оперировать в программе, описывающей *E*-сеть, являются:

`finish` — модельное время, при котором прекращается имитация, тип long;

`clock` — текущее модельное время, только для чтения, тип long;

`r` — значение разрешающей позиции текущего перехода, тип int;

`d` — значение временной задержки текущего перехода, тип long;

`s[i].atr[j]` — *j*-й атрибут *i*-й простой позиции, тип long;

`q[i].last->atr[j]` — *j*-й атрибут последней фишки в *i*-й позиции-очереди, тип long;

`q[i].first->atr[j]` — *j*-й атрибут первой фишки в *i*-й позиции-очереди, тип long;

`q[i].nmark` — количество фишек в *i*-й позиции-очереди, только для чтения, тип int;

`s[i].nmark` — если равен 1, простая позиция с номером *i* содержит фишку, если 0, то простая позиция с номером *i* свободна, только для чтения, тип int;

`q[i].strategy` — порядок обслуживания фишек в очереди:

`F` — обслуживание по принципу FIFO (по умолчанию);

`L` — обслуживание по принципу LIFO, тип char.

Кроме системных переменных пользователь может воспользоваться системными функциями-датчиками случайных чисел:

rnd(n) — функция генерации псевдослучайных чисел равномерно распределенных на отрезке **[0, n]**, **n** — положительный аргумент типа int;

expon(n) — функция генерации псевдослучайных чисел, распределенных по экспоненциальному закону с математическим ожиданием **n**, **n** — положительный аргумент типа int.

При каждом прогоне модели датчики случайных чисел автоматически инициализируются, поэтому результаты прогонов одной модели, содержащей обращения к этим функциям, различны, что дает возможность накапливать статистику поведения исследуемой системы.

Разделы имеют следующие функциональные назначения и имена:

A: — раздел описаний переменных и функций пользователя. Эти переменные и функции могут использоваться в любых разделах программы, кроме раздела **S**. Их имена должны начинаться со знака подчеркивания «_», а во всем остальном описания полностью эквивалентны описанию переменных и функций языка Си. Никаких ограничений на количество и содержание тел функций не накладывается. Пользователь может в этом разделе использовать директивы компилятора Си **#include**, **#define** и др.;

I: — раздел определения начального состояния модели. Этот раздел можно рассматривать как тело функции, которая исполняется непосредственно перед началом процесса имитации, поэтому в него можно включить операторы по заданию начальных значений системных и пользовательских переменных модели: число шагов моделирования (finish), значения атрибутов фишек в помеченных ими позициях и т. п. В разделе возможно использование любых операторов и функций языка Си;

R: — раздел описания процедур определения значений разрешающих позиций. В этом разделе для переходов типа «переключатель» и «выборка» определяется значение системной переменной **r**. Это значение должно быть типа int и лежать в интервале от 1 до **n**, где **n** — число выходных (для переключателя) или входных (для выборки) позиций перехода. Номер определяемого перехода **n** указывается с помощью метки вида **Tn:**. Текст программы, следующий за такой меткой и до следующей метки или конца раздела, является телом процедуры определения значения разрешающей позиции данного перехода. Тут возможно использование любых операторов и функций языка Си, среди которых должен быть оператор присваивания значения системной

переменной `r`. Если переход указанного типа не описан в этом разделе, то для определения значения разрешающей позиции используется стандартная процедура: определяется наименьший номер среди свободных выходных позиций (для переключателя) или занятых позиций (для выборки);

`D:` — раздел описания процедур вычисления временных задержек срабатывания переходов. В этом разделе для переходов, имеющих ненулевую задержку, определяется значение системной переменной `d`. Это значение должно быть типа long и положительно. Номер определяемого перехода n указывается с помощью метки вида `Tn:`. Текст программы, следующий за такой меткой и до следующей метки или конца раздела, является телом процедуры определения временной задержки данного перехода. Тут возможно использование любых операторов и функций языка Си, среди которых должен быть оператор присваивания значения системной переменной `d`. Если переход не описан в этом разделе, то временная задержка для него устанавливается равной нулю;

`T:` — раздел описания процедур преобразования атрибутов фишек при срабатывании переходов. Номер определяемого перехода `n` указывается с помощью метки вида `Tn:`. Текст программы, следующий за такой меткой и до следующей метки или конца раздела, является телом процедуры преобразования атрибутов фишек, проходящих через данный переход. Тут возможно использование любых операторов и функций языка Си. Доступ к атрибутам осуществляется с помощью системных переменных вида `s[i].atr[j]`. Если переход не описан в этом разделе, то преобразования атрибутов не происходит;

`S:` — раздел определения элементов *E*-сети, для которых проводится сбор статистики. В данном разделе разрешается использование только системных функций сбора статистики языка EDEF. Эти функции не возвращают никаких значений и позволяют накапливать статистику стандартного вида по различным объектам *E*-сети. По окончании процесса моделирования вся накопленная статистика выводится на экран;

`E:` — раздел определения процедуры, которая исполняется по окончании процесса моделирования. Этот раздел можно рассматривать как тело функции, которая исполняется непосредственно после завершения процесса имитации, поэтому, например, в него можно включить операторы по обработке переменных пользователя, принимавших участие в работе модели, получению на их основе результатов моделирования и любые другие операторы и функции языка Си.

Пример 3.4. Моделирование парикмахерской в программе EVA

E-сеть, моделирующая функционирование парикмахерской, может иметь вид, приведенный на рис. 3.13. Структура этой *E*-сети (ее изображение) задается с помощью графического редактора EEE. Каждая фишка в сети имеет один атрибут, отвечающий за пол клиента (1 — мужчина, 2 — женщина). Переход *T*1 генерирует поступление фишек (клиентов) в парикмахерскую и при этом устанавливается значение их атрибута (1 или 2). Разрешающая позиция *R*1 перехода *T*2 определяет по этому атрибуту, в какую из очередей (*Q*1 или *Q*2) поместить клиента (очередь *Q*1 — в мужской зал, очередь *Q*2 — в женский). Через переходы *T*3 и *T*4 фишки из очередей поступают в первую из свободных выходных позиций *P*2–*P*7 этих переходов (на обслуживание к мастерам). Срабатывание переходов *T*5–*T*10 означает, что соответствующий посетитель обслужен и покидает парикмахерскую.

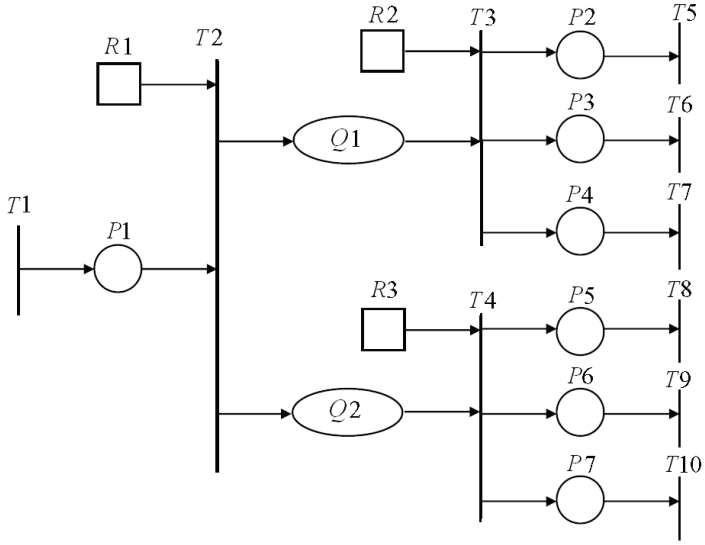


Рис. 3.13. *E*-сеть, моделирующая функционирование парикмахерской

Временные задержки срабатывания переходов *T*5–*T*10 соответствуют длительности обслуживания клиентов соответствующими мастерами, перехода *T*1 — интервалу времени, через который поступают клиенты в парикмахерскую.

Программа, описывающая функционирование этой *E*-сети на языке EDEF, имеет следующий вид:

```
A: int _Men = 0,_Women = 0; /* Суммарные количества обслуженных */
/* мужчин и женщин */
I: fprintf(stderr,»Enter Finish: «);
scanf(«%ld», &finish);
R:
T2: r = s[1].atr[1]; /* Если атрибут равен 1, то пришел */
/* мужчина и он ставится в очередь Q1*/
/* Если атрибут равен 2, то пришла */
/* женщина, и она ставится в очередь Q2*/
T:
T1: if (rnd(100) > 50) s[1].atr[1] = 1L; /*Задание пола */
else s[1].atr[1] = 2L; /*клиента */
T3: _Men++; /* Подсчет общего числа обслуженных */
T4: _Women++; /* мужчин и женщин */
D:
T1: d = expon(7); /* Интервал времени между приходом */
/* клиентов в парикмахерскую, пуассоновский поток */
T5: d = 10 + rnd(15); /* Временные интервалы обслуживания */
T6: d = 10 + rnd(20); /* мужчин для каждого из парикмахеров */
T7: d = 10 + rnd(25);
T8: d = 20 + rnd(20); /* Временные интервалы обслуживания */
T9: d = 20 + rnd(45); /* женщин для каждого из парикмахеров */
T10: d = 20 + rnd(60);
S:
stat_q(1,1,1,10); /* Накопление статистики по очередям */
stat_q(2,1,1,15); /* Q1 и Q2 */
E: /* Вывод на экран общего числа обслуженных мужчин и женщин */
printf(«\n\n\n\n\n»);
printf(« Men = %d Women = %d\n», _Men, _Women);
```

В разделе `A:` описаны две переменных пользователя `_Men` и `_Women`, которые являются счетчиками обслуженных мужчин и женщин.

В разделе `I:` запрашивается ввод с клавиатуры значения системной переменной `finish` (число шагов работы модели). Можно условно считать один шаг модельного времени равным одной минуте реального времени. Поэтому, если задать значение этой переменой равным 600, то это означает, что будет осуществляться моделирование десяти часов работы парикмахерской.

В разделе `R:` определена разрешающая процедура перехода `T2:` — переместить фишку из позиции *P*1 по первой сверху выходной дуге перехода в очередь *Q*1, если ее атрибут равен 1 (мужчина), или по второй дуге в очередь *Q*2, если атрибут равен 2 (женщина). Остальные разрешающие процедуры для переходов `T3` и `T4` выбраны по умолчанию, следовательно, фишка перемещается в первую сверху свободную позицию.

В разделе `T:` определены процедуры преобразования атрибутов для переходов: `T1` — устанавливается значение атрибута фишки в позиции *P*1 на 1 или 2 (задание пола клиента);

`T3` — фактически атрибут не изменяется, но при срабатывании перехода *T*3 счетчик обслуженных мужчин увеличивается на единицу;

`T4` — фактически атрибут не изменяется, но при срабатывании перехода *T*4 счетчик обслуженных женщин увеличивается на единицу.

Процедуры преобразования атрибутов для остальных переходов не определены, а значит, атрибуты фишек при переходе через них не изменяются.

В разделе `D:` определены временные задержки срабатывания переходов `T1` и с `T5` по `T10`, выраженные в шагах модельного времени, а следовательно, в минутах работы парикмахерской. Временные задержки для остальных переходов не определены, что эквивалентно их равенству нулю.

В разделе `S:` используются системные функции накопления статистики для очередей `Q1` и `Q2.` При этом в гистограммах для каждой из длин очередей подсчитывается общее время, в течение которого данная длина сохранялась. Для очереди `Q1` (мужчин) максимальная регистрируемая длина равна 10, а для очереди `Q2` (женщин) — 15.

В разделе `E:` определены действия системы по окончанию моделирования (перед выводом статистики на экран). Выводится на экран количество обслуженных за период моделирования мужчин и женщин.

В заключение главы отметим, что могут быть и другие подходы к моделированию *E*-сетей. Например, *E*-сетевая модель может быть построена на основе принципов функциональной декомпозиции, реализующей концепцию построения сложной системы из набора простых подсистем. Поэтому модели, разработанные на основе *E*-сетей, могут быть формально транслированы в классические сети Петри, что позволяет использовать формальные методы их анализа, например,

с помощью матриц [31]. При этом технология моделирования систем в виде *E*-сетей может быть реализована с помощью системы моделирования MATLAB.

Вопросы и задания к главе 3

1. Поясните связь декомпозиции сложной системы с формированием ее модели в виде *E*-сети.
2. Каковы должны быть свойства объекта (как результата декомпозиции сложной системы) для его представления в виде объектов *E*-сетевого моделирования?
3. В чем состоит отличие *E*-сетей от примитивных сетей Петри? Имеют ли эти отличия принципиальный характер?
4. Какое из утверждений — «*E*-сети есть расширение сетей Петри» и «*E*-сети есть ограниченные сети Петри» — ближе к истине?
5. Назовите сходства и различия двух видов позиций *E*-сетей — безусловных позиций и решающих позиций.
6. Какова роль макропозиций в *E*-сетях? В чем состоит их отличие от макропозиций в сетях Петри?
7. Дайте формальное определение *E*-сети.
8. Какие процедуры обработки информации могут выполняться в переходах *E*-сети?
9. Какой безусловный переход может моделировать занятость некоторого устройства (объекта) в течение определенного времени?
10. Назовите разрешительные переходы в *E*-сети. Какими сигналами они управляются и какие функции реализуют?
11. Поясните работу макропозиции *Генератор*. При каком условии эта макропозиция может генерировать пуассоновский поток меток (событий)?
12. Можно ли назвать макропозицию *Аккумулятор* «терминатором» некоторой ветви *E*-сети?
13. На рис. 3.8 изображена модель мультипрограммной вычислительной системы в формализме *E*-сети. Переходы $T1$ и $T3$ имеют разрешительные процедуры $R1$ и $R2$ соответственно. Можно ли от последних избавиться, не нарушая работу модели? Каким образом это сделать?

14. Поясните, с помощью каких элементов *E*-сети можно в ней хранить и передавать транзакты? Поясните на примере задержку транзакта на конкретное число временных тактов.
15. Каково содержательное назначение обратной связи с перехода T6 на безусловную позицию S6 в *E*-сети на рис. 3.10?
16. Поясните вставку в *E*-сети на рис. 3.10 безусловного перехода S7 и всего последующего фрагмента сети.
17. Каким образом выглядела бы *E*-сеть параллельной системы, одним из компонентов которой была бы *E*-сеть на рис. 3.11?
18. Почему в практике моделирования сложных систем отсутствует универсальный программный инструмент для создания и исследования моделей на основе *E*-сетей?
19. После ознакомления с системой моделирования EVA какие ее модули можно использовать во вновь разрабатываемых программных инструментах, а какие следует признать устаревшими?
20. Одним из главных недостатков системы EVA следует признать обязательное знакомство пользователя с языком программирования Си. Каким образом следовало бы преодолеть этот недостаток во вновь разрабатываемых программах моделирования *E*-сетей?

4. Сортирующие сети

В системах обработки данных значительную долю времени занимают операции поиска и сортировки, причем при организации поиска, как правило, используют сортировку признаков (ключей, ссылок, меток и др.) данных. Эффективность их выполнения во многом может определять эффективность всей системы, иногда и ее работоспособность. Поэтому естественен пристальный интерес к теории и практике этих операций, выразившийся, например, в виде третьего тома известной монографии Д. Кнута [32], в которой автор на рубеже середины 70-х годов прошлого века провел обобщенный анализ многочисленных работ в этой области. В ней представлен подробный анализ алгоритмов сортировки, один из которых непосредственно связан с сетями. Речь идет о так называемых сетях сортировки, в которых последовательность сравнений в наименьшей степени зависит от предыстории, обеспечивая тем самым возможность параллельной обработки данных.

Сортирующие сетевые алгоритмы используют в отличие от стандартных ЭВМ другую физическую базу — сеть компараторов. Различие названных аппаратных поддержек алгоритмов весьма существенно. Во-первых, в процессе работы компараторы могут выполнять только операции сравнения. Следовательно, реализовать на них алгоритмы, подобные сортировке подсчетом, невозможно. Во-вторых, ЭВМ работают последовательно, выполняя одну операцию за один такт работы. Сеть компараторов может работать параллельно, т. е. выполнять одновременно несколько операций, благодаря чему можно отсортировать n чисел за время существенно меньшее n тактов.

Ниже представлены основные результаты главы 28 капитальной монографии [33], дающие достаточно полное представление о структуре, процессах обработки данных и возможных приложениях сортирующих сетей. Большую роль в выборе темы главы сыграло методически выверенное изложение материала и сопутствующий список вопросов и заданий в [33].

4.1. Сети компараторов

Сортирующие сети строятся из компараторов, соединенных проводами. Компаратор (*comparator*) имеет два входа x, y и два выхода x', y' (рис. 4.1, *а*). Компаратор получает на вход два числа и переставляет их, если они идут в неправильном порядке. Другими словами,

$$\begin{cases} x' = \min(x, y), \\ y' = \max(x, y). \end{cases} \tag{4.1}$$

Договоримся схематично изображать компаратор в виде вертикального отрезка, как на рис. 4.1, *б*. Входы расположены слева, а выходы — справа, при этом верхний выход соответствует минимальному числу, а нижний — максимальному числу.

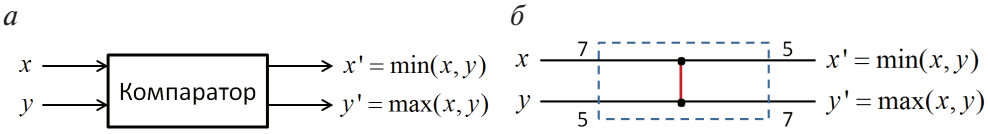


Рис. 4.1. Компаратор со входами x, y и выходами x', y' — *а*, и тот же компаратор в виде вертикальной линии — *б*

Считаем, что время работы компаратора (между получением входных данных и выдачей выходных) постоянно и одинаково для всех компараторов. С помощью проводов выходы одних компараторов соединяют со входами других. Кроме того, сеть имеет входные и выходные провода.

Рассмотрим сеть компараторов с n входными проводами $a_1, a_2, \ldots, a_n$ и n выходными проводами $b_1, b_2, \ldots, b_n$. На вход поступает последовательность n чисел (которые мы будем обозначать $[a_1, a_2, \ldots, a_n]$, как и сами провода); результатом работы будет последовательность $[b_1, b_2, \ldots, b_n]$.

На рис. 4.2 приводится пример сети компараторов. Здесь сеть с n входами и n выходами изображается n горизонтальными прямыми, которые в некоторых местах соединены вертикальными отрезками — компараторами. Горизонтальная прямая — это графический образ, на самом деле это не один провод, а несколько реальных проводов. Например, верхняя прямая на рис. 4.2 состоит из трех проводов: входного провода a_1, соединенного со входом компаратора A; провода, со-

единяющего верхний выход компаратора A со входом компаратора C, а также выходного провода B, соединенного с верхним выходом компаратора C. Поэтому одиночными проводами являются только участки между компараторами. На рис. 4.2, *а* показаны входные значения в момент времени 0, на рис. 4.2, *б* — значения на выходах компараторов A и B в момент времени 1. В момент времени 2 появляются выходные значения компараторов C и D (рис. 4.2, *в*), при этом выходные значения b_2, b_3 и b_4 еще не найдены. Они определяются на третьем шаге на выходах компаратора E (рис. 4.2, *г*).

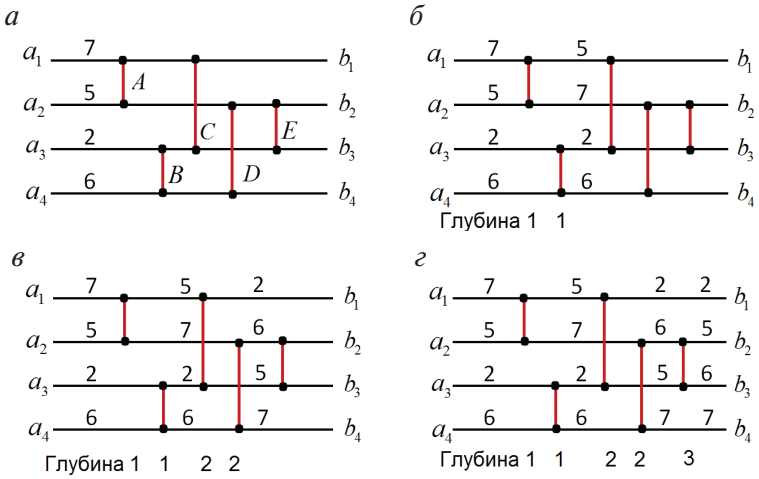


Рис. 4.2. Сеть компараторов с 4 входами и 4 выходами в момент времени 0 — *а*. Показаны выходы компараторов A и B в момент времени 1 — *б*, выходы компараторов C и D в момент времени 2 — *в* и выход сети после работы компаратора E в заключительный момент времени 3 — *г*

Любая схема описанного вида (из прямых и отрезков) задает схему соединения компараторов. В этой схеме каждый вход любого из компараторов соединен либо с одним из входных проводов a_1, a_2, ..., a_n, либо с выходом другого компаратора. Каждый выход любого из компараторов соединен либо с входом другого (ровно одного), либо с одним из выходных проводов b_1, b_2, ..., b_n. При этом нет циклов: идя по проводам и проходя компараторы от входов к выходам, нельзя вернуться к исходному компаратору. Поэтому любая схема из компараторов, обладающая только что указанными свойствами, может быть изображена в виде прямых — путей сигналов от входов к выходам — и отрезков — моделей, встречающихся на этих путях компараторов.

Считается, что каждый компаратор выдает выходные значения через единицу времени после того, как получит входные значения. Рассмотрим сеть рис. 4.2, *а* и представим себе, что в момент времени 0 на вход поступают числа [7, 5, 2, 6]. В этот момент компараторы *A* и *B* (и только они) получают входные данные и начинают работать параллельно. Поэтому в момент времени 1 на их выходах появятся числа (см. рис. 4.2, *б*), которые позволят начать параллельную работу компараторам *C* и *D*, но не компаратору *E*. Компаратору *E* придется дожидаться момента времени 2, когда *C* и *D* закончат работу (см. рис. 4.2, *в*). Еще через единицу времени выходная последовательность (2, 5, 6, 7) будет полностью готова, хотя одно из четырех значений было готово раньше.

Как видно из этого примера, время работы сети, прошедшее с момента получения входной последовательности до выдачи выходной последовательности, определяется максимальным числом компараторов, стоящих на пути от входа к выходу. Можно определить глубину сети следующим образом: входные провода имеют нулевую глубину, глубина выходов компаратора (подключенных к нему проводов) равна max (d_x, d_y), где d_x и d_y — глубины его входов (подключенных к ним проводов). Это определение корректно, поскольку нет циклов. Назовем *глубиной компаратора* глубину выходящих из него проводов, а *глубиной сети* — максимальную глубину выходных проводов или, что то же самое, максимальную глубину ее компараторов. Например, глубина сети на рис. 4.2 равна 3 (компаратор *E* и его глубина 3). Глубина компаратора равна времени, которое пройдет от начала работы по появления ответов на его выходах, так что *время работы сети в целом равно ее глубине*. *Размером* сети называют число компараторов в ней.

Любая сеть компараторов как-то переставляет числа, поданные на ее входы. Ее называют *сортирующей сетью*, если для любой входной последовательности получающаяся из нее выходная последовательность монотонно возрастает: $b_1 \le b_2 \le \ldots \le b_n$.

Конечно, далеко не каждая сеть — сортирующая, но сеть на рис. 4.2 таковой является. В самом деле, в момент времени 1 минимальный из входов находится на верхнем выходе одного из компараторов *A* и *B*, и в момент времени 2 он окажется на верхнем выходе компаратора *C*. Аналогичным образом максимальный из входов окажется в этот момент на нижнем выходе компаратора *D*. Оставшиеся два числа упорядочиваются компаратором *E* в момент времени 3.

Сети компараторов аналогичны алгоритмам сортировки, однако для каждого n необходимо строить свою сеть, в то время как один и тот же алгоритм сортировки может работать для последовательностей произвольной длины. В этой главе будет построено семейство `sorter` эффективных сортирующих сетей — для каждого n в нем будет своя сеть, которую обозначим `sorter(n)`.

4.2. Правило нуля и единицы

Из правила нуля и единицы следует, что если сеть компараторов упорядочивает любую последовательность нулей и единиц, то она является сортирующей и упорядочивает любую последовательность чисел — целых, вещественных, или вообще элементов произвольного линейно упорядоченного множества. Тем самым, доказательство того, что построенная сеть является сортирующей, можно свести к воздействию на ее вход всего лишь вектора из нулей и единиц.

Доказательство [33] использует понятие монотонно возрастающей функции.

Теорема. Если сеть компараторов преобразует последовательность $\mathbf{a} = [a_1, a_2, ..., a_n]$ в последовательность $\mathbf{b} = [b_1, b_2, ..., b_n]$, а f — монотонно возрастающая функция, то, подавая на вход сети последовательность $\mathbf{f(a)} = [f(a_1), f(a_2), ..., f(a_n)]$, получим на выходе $\mathbf{f(b)} = [f(b_1), f(b_2), ..., f(b_n)]$.

Проверим это сначала для одного компаратора, т. е. покажем, что компаратор, получив на вход $f(x)$ и $f(y)$, выдаст на выходах $f(\min(x,y))$ *и* $f(\max(x,y))$ (рис. 4.3).

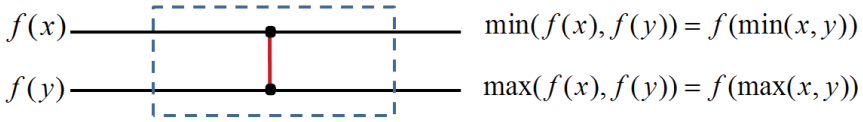


Рис. 4.3. Компаратор и монотонная функция

По определению, на верхнем выходе будет $f(\min(x,y))$, а на нижнем — $f(\max(x,y))$. Остается заметить, что для возрастающей функции f из $x \le y$ следует $f(x) \le f(y)$, поэтому

$$\min(f(x), f(y)) = f(\min(x, y)),$$
$$\max(f(x), f(y)) = f(\max(x, y)).$$

На рис. 4.4 показана сеть без применения ко входам монотонной функции $f(x) = \lceil x/2 \rceil$ (округление в сторону плюс бесконечности) и с применением этой функции. Сортировка выполнена этими сетями одинаково.

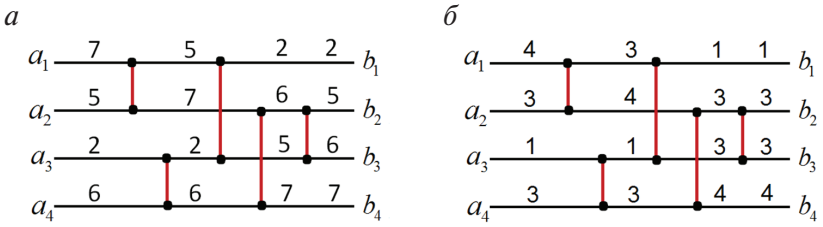


Рис. 4.4. Сортирующая сеть на рис. 4.2 для входов (7, 5, 2, 6) без — *а* с применением монотонной функции $f(x) = \lceil x/2 \rceil$ ко всем входам — *б*

Другими словами, если применить ко входным значениям компаратора x и y монотонную функцию f, с его выходными значениями произойдет то же самое.

Это свойство верно не только для одного компаратора, но и для любой сети компараторов. Пусть на входы сети поданы значения $\mathbf{a} = [a_1, a_2, ..., a_n]$. Через некоторое время на всех проводах (в том числе выходных) установятся некоторые значения. Теперь заменим входы на $\mathbf{f(a)} = [f(a_1), f(a_2), ..., f(a_n)]$. Выходы компараторов глубиной 1 также заменятся на результат применения к ним функции f. Следовательно, с выходами компараторов глубиной 2 произойдет то же самое, и так далее. Рассуждая по индукции, мы видим, что на выходах схемы появятся значения $\mathbf{f(b)} = [f(b_1), f(b_2), ..., f(b_n)]$, где $\mathbf{b} = [b_1, b_2, ..., b_n]$ — прежние выходные значения.

Рис. 4.4 иллюстрирует этот вывод для сортирующей сети на рис. 4.2 и функции $f(x) = \lceil x/2 \rceil$. Часть (рис. 4.4, *а*) показывает значения на проводах до применения f (дублируя рис. 4.2, *г*), а часть (см. рис. 4.2, *б*) — после.

Правило нуля и единицы. Если сеть компараторов с n входами правильно упорядочивает все 2^n возможных последовательностей нулей и единиц, то она является сортирующей, т. е. правильно упорядочивает любую числовую последовательность (см. рис. 4.5).

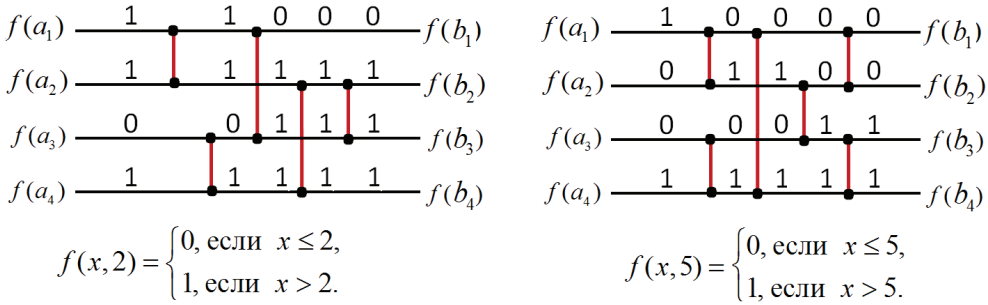


Рис. 4.5. Сортирующие сети для входов (7, 5, 2, 6)

Пусть это не так, и есть числовая последовательность $a_1, a_2, ..., a_n$, на которой сеть ошибается. Это означает, что есть элементы a_i, a_j, для которых $a_i < a_j$ и a_j попадает в выходную последовательность раньше a_i. Нам надо показать, что существует последовательность нулей и единиц, на которой сеть работает неправильно. Для этого рассмотрим монотонную функцию *f*:

$$f(x)=\begin{cases}0, \text{ если } x\le a_i,\\ 1, \text{ если } x>a_i.\end{cases}$$

Применим ее к входам сети, т. е. подадим на вход последовательность нулей и единиц $f(a_1), f(a_2), ..., f(a_n)$. К выходным **значениям** также применится функция *f*. При этом $f(a_j)$ будет стоять на месте a_j, т. е. раньше $f(a_i)$. Но $f(a_j)=1$, а $f(a_i)=0$. Противоречие доказывает правило нуля и единицы.

4.3. Битонический сортировщик

Построение эффективной сортирующей сети начнем с так называемого *битонического* сортировщика, который сортирует так называемые битонические последовательности. *Битонической* назовем любую последовательность, которая сначала возрастает, а потом убывает, или получается из таковой циклическим сдвигом. Если записать элементы битонической последовательности по кругу, то минимальный и максимальный ее элементы делят последовательность на два монотонных участка. (Отсюда и название: *би* — два, *тон* — моно*тон*ный).

Если входная последовательность является битонической последовательностью нулей и единиц, то после ее обработки любой элемент верхней половины выхода меньше любого элемента нижней половины (или равен ему); одна из половин — битоническая, а другая состоит только из нулей или только из единиц («чистая»).

Например, следующие последовательности (1,4,6,8,3,2), (6,9,4,2,3,5) и (9,8,3,2,4,6) — битонические. Битонические последовательности нулей и единиц имеют вид либо $1^i0^j1^k$, либо $0^i1^j0^k$, где $i, j, k \geq 0$. Записанные по кругу, они состоят из двух групп — в одной нули, в другой единицы, и группы не смешиваются. Отметим, что монотонные последовательности являются частным случаем битонических.

В этом разделе будет построен битонический сортировщик — сеть компараторов, правильно сортирующая битонические последовательности нулей и единиц.

Полуочиститель. Битонический сортировщик состоит из нескольких частей разных размеров, которые будем называть «полуочистителями». Полуочиститель размером n есть сеть глубиной 1, в которой компараторы соединяют провода одной половины с проводами другой (i и $i + n/2$ соединены при $i = 1, 2, \ldots, n/2$; предполагается, что n четно). На рис. 4.6 показан `Clean12` [8] — полуочиститель размером 8.

Основное свойство полуочистителя, объясняющее его название, состоит в следующем. Пусть на вход полуочистителя подана битоническая последовательность, состоящая из нулей и единиц. Получающаяся выходная последовательность также состоит из нулей и единиц и обладает свойствами:

- ее верхняя и нижняя половины — битонические;
- любой элемент верхней половины меньше любого элемента нижней или равен ему;
- хотя бы одна из половин — монотонная или (для обобщения названия) чистая битоническая.

Для доказательства предположим, что вход имеет вид 00... 011 ... 100... 0 (случай 11 ... 100 ... 011 ... 1 симметричен). Сеть `Clean12(n)` сравнивает вход a_i со входом $a_{i+n/2}$, поэтому возможны три расположения блока единиц относительно середины последовательности, причем случай, когда середина приходится на блок единиц, распадается на два подслучая. Как видно из рис. 4.7, утверждение справедливо во всех четырех случаях. Нулевые участки на рисунке белые, единичные — серые. Входная последовательность разрезается на две половины, ко-

торые почленно сравниваются. Случаи, когда точка раздела попадает в кусок из единиц, показаны на рис. 4.7, *а*, *б*, а когда проходит по нулевому участку — на рис. 4.7, *в*, *г*.

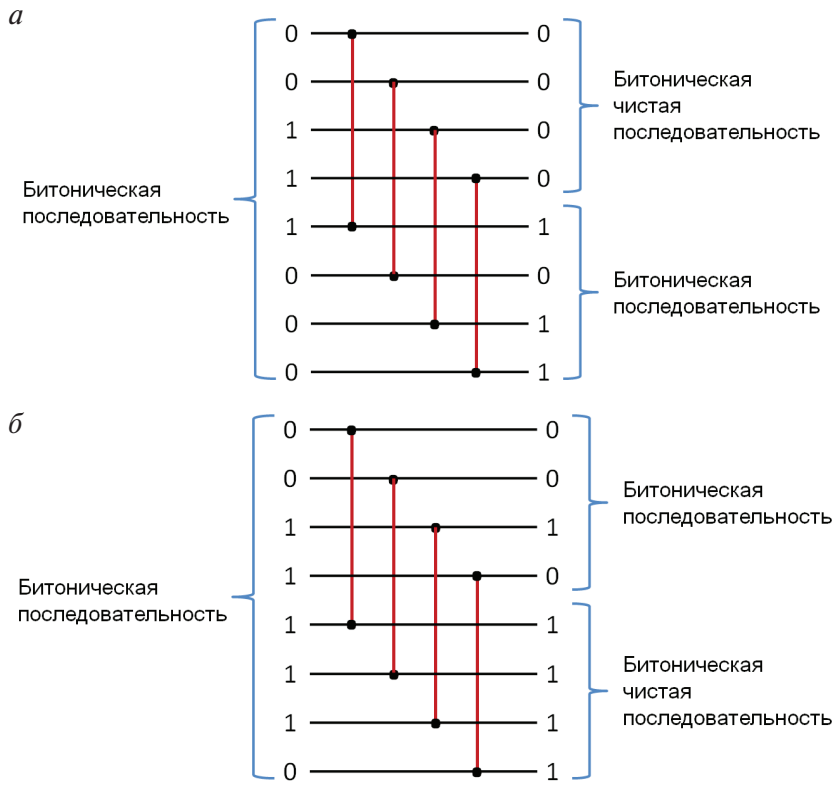


Рис. 4.6. Полуочиститель `Clean12` [8]
и две различные входные последовательности нулей и единиц

Битонический сортировщик рекурсивно строится из полуочистителей, как показано на рис. 4.8. Сеть `BitonSort(n)` состоит из полуочистителя `Clean12(n)` и двух экземпляров сети `BitonSort(n/2)`. Полуочиститель делает из входной битонической последовательности две битонические последовательности (одна из них чистая) половинного размера. При этом любой элемент верхней меньше или равен любого элемента нижней. После этого остается упорядочить каждую из них при помощи сети `BitonSort(n/2)`. На рис. 4.8, *а* показана общая структура сети: после блока `Clean12(n)` идут две копии блока `BitonSort(n/2)`, работающие параллельно. На рис. 4.8, *б* рекурсия развернута: серым цветом

показаны блоки `Clean12`. Один из возможных вариантов сортируемых значений указан на проводах. Таким образом строится битонический сортировщик с n входами, где n — произвольная степень двойки.

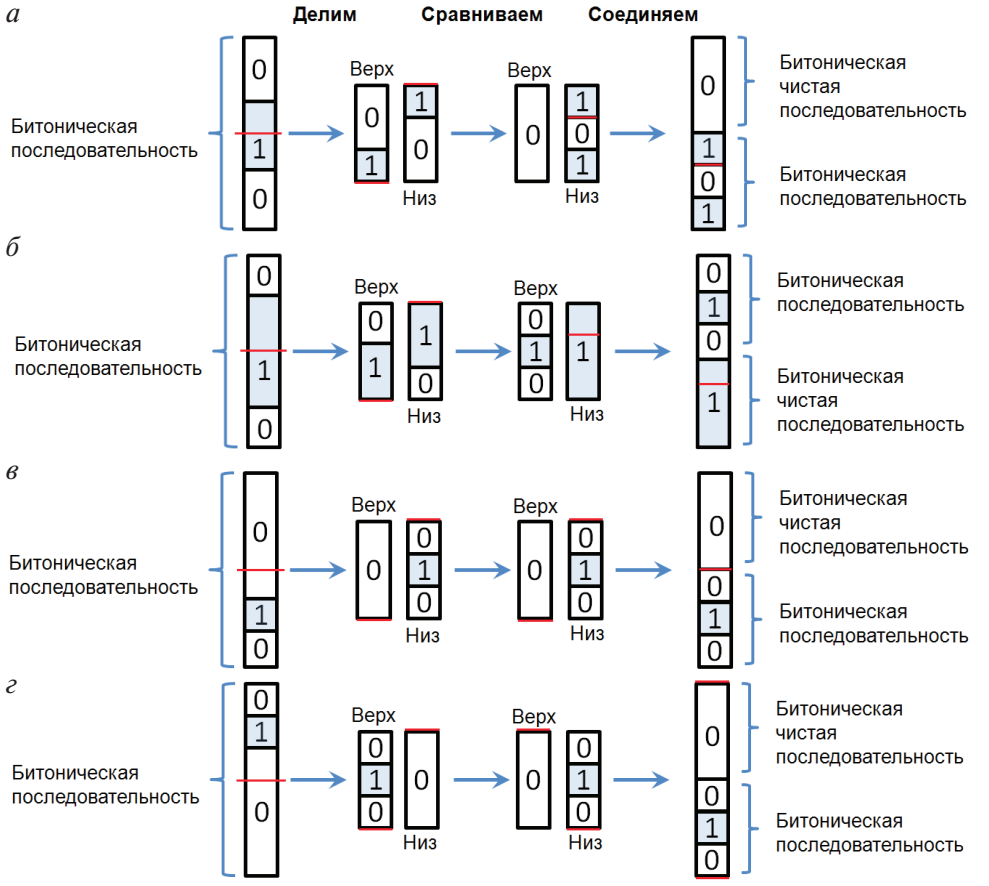


Рис. 4.7. Четыре возможных варианта при работе сети `Clean12(n)`. Два варианта — точка раздела попадает в участок из единиц — *а*, *б* и два варианта — точка раздела попадает на нулевой участок — *в*, *г*

Глубина $D(n)$ сети `BitonSort(n)` дается соотношением

$$D(n)=\begin{cases}0, & \text{если } n=1,\\ D\left(\frac{n}{2}\right)+1, & \text{если } n=2k \text{ и } k\geq 1.\end{cases} \qquad (4.2)$$

из которого видно, что $D(n)=\log_2(n)$.

Итак, битонические последовательности нулей и единиц сортируются правильно, отсюда следует правило нуля и единицы для битонических последовательностей, и это значит, что любые битонические последовательности в таком сортировщике сортируются правильно.

а *б*

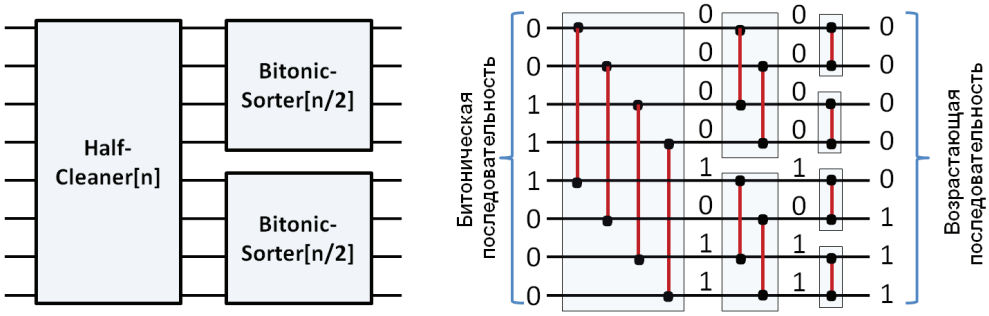


Рис. 4.8. Сортирующая сеть `BitonSort(n)` при $n = 8$: структурная схема — *а* и один из возможных вариантов сортировки битонической последовательности — *б*

4.4. Сливающая сеть

Сливающей сетью называется сеть компараторов, соединяющая две упорядоченные последовательности одинаковой длины в одну упорядоченную последовательность двойной длины. Такая сеть под названием `Merger(n)` легко получается модификацией сети `BitonSort(n)`.

Если записать вторую последовательность в обратном порядке, т. е. в порядке убывания, и приписать ее к концу первой, отсортированной в порядке возрастания, то в результате слияния таких двух упорядоченных последовательностей получится битоническая последовательность, которую уже можно упорядочить с помощью битонического сортировщика. Например, для объединения $X = 00000111$ и $Y = 00001111$ мы приписываем к X перевернутую последовательность $Y^R = 11110000$ и получаем битоническую последовательность $XY^R = 0000011111110000$. Осталось применить к XY^R битонический сортировщик.

Следуя этому алгоритму, для построения сети `Merger(n)`, сливающей последовательности $a_1, a_2, \ldots, a_{n/2}$ и $a_{n/2+1}, \ldots, a_n$, необходимо так перестроить первый полуочиститель сети `BitonSort(n)`, чтобы добиться эффекта «переворачивания» второй последовательности. В обычном

полуочистителе вход a_i сравнивается со входом $a_{n/2+1}$ (при $i = 1,2,...,n/2$), поэтому теперь будем сравнивать a_i со входом a_{n-i+1}. На рис. 4.9 показан перестроенный полуочиститель, в сравнении с обычным разница состоит в том, что провода в нижней его половине переставлены в обратном порядке. При этом верхняя и нижняя половины выходов по-прежнему обладают свойствами, указанными для битонической последовательности, которая, записанная в обратном порядке, остается битонической.

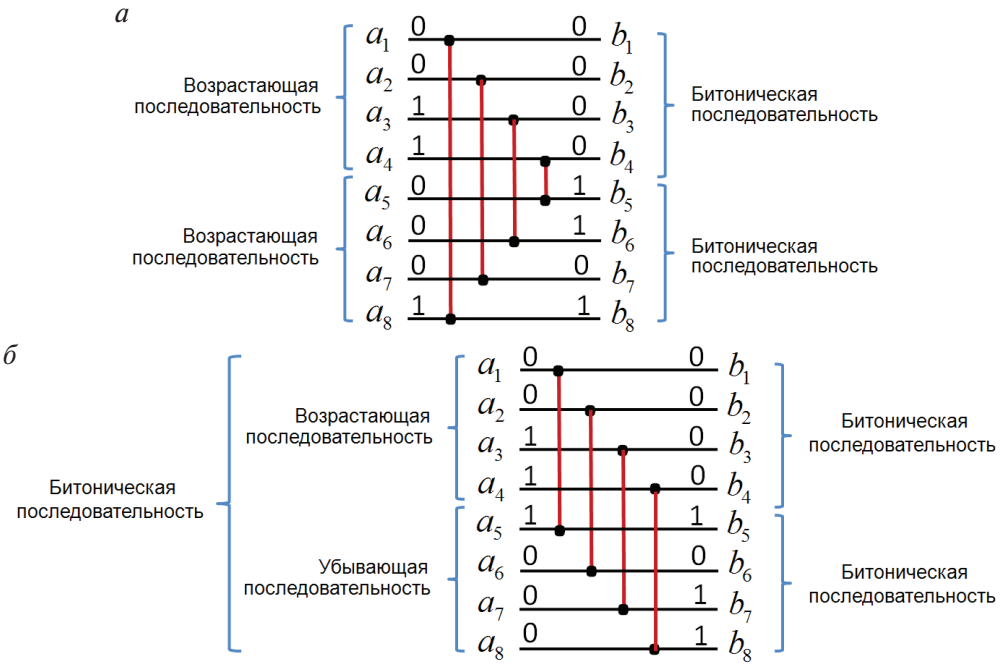


Рис. 4.9. Входная часть сети `Merger(8)` — *а* отличается от сети `Clean12(8)` — *б* тем, что нижние 4 провода перевернуты

На рис. 4.9, *а* входная часть сети **`Merger`** преобразует две возрастающие последовательности — $\langle a_1, a_2, ..., a_{n/2} \rangle$ и $\langle a_{n/2+1}, a_{n/2+2}, ..., a_n \rangle$ — в две битонические последовательности: «чистую» $\langle b_1, b_2, ..., b_{n/2} \rangle$ и стандартную $\langle b_{n/2+1}, b_{n/2+2}, ..., b_n \rangle$. В полуочистителе (рис. 4.9, *б*) две входные последовательности — возрастающая $\langle a_1, a_2, ..., a_{n/2} \rangle$ и убывающая $\langle a_{n/2+1}, a_{n/2+2}, ..., a_n \rangle$ — образуют одну битоническую последовательность,

которая преобразуется в две аналогичные битонические последовательности: «чистую» $\langle b_1, b_2, ..., b_{n/2} \rangle$ и стандартную $\langle b_{n/2+1}, b_{n/2+2}, ..., b_n \rangle$.

Для завершения слияния остается упорядочить обе половины при помощи двух копий сети `BitonSort(n)`. Полученная сеть `Merger(n)` показана на рис. 4.10. Ее глубина такая же, как у сети `BitonSort(n)`, т. е. $\log_2(n)$.

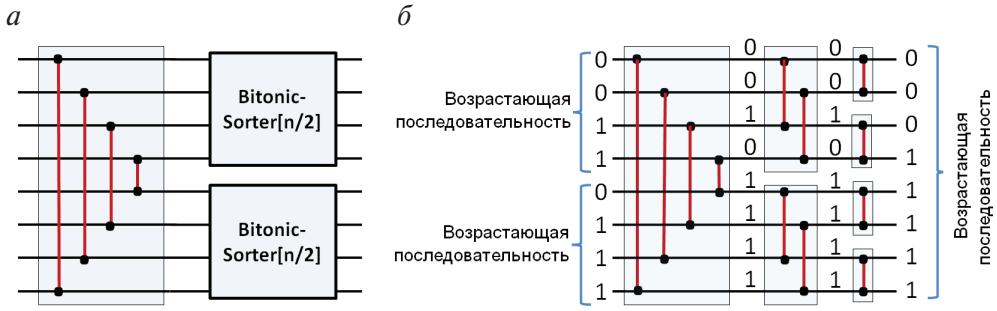


Рис. 4.10. Общая структура сортировочной сети — *а* и ее полная схема — *б*

На рис. 4.10, *а* сливающая сеть `Merger` соединяет две отсортированные последовательности в одну. Структура сети `Merger` совпадает со структурой сортирующей сети `BitonSort` (см. рис. 4.8, *а*) за исключением первого блока: вместо полуочистителя `HalfCleaner[n]` здесь стоит `Merger(n)`, т. е. она получается из сети `BitonSort` модификацией первого каскада: теперь сравниваются *i*-е с начала и конца элементы. Общая структура сети: первый каскад, затем два экземпляра сети `BitonSort(n/2)`. Полная схема сливающей сети при `n=8` показана на рис. 4.10, *б* — пример слияния двух возрастающих последовательностей.

4.5. Сортирующая сеть

Теперь все готово для построения сортирующей сети `Sorter(n)`, реализующей алгоритм сортировки слиянием. Устройство сети показано на рис. 4.11.

Сеть `Sorter(n)` строится рекурсивно: две половины входной последовательности упорядочиваются сетями `Sorter(n/2)`, а затем соединя-

ются при помощи сети `Merger(n)`. В качестве `Merger(2)` используется компаратор. Схема построения показана на рис. 4.11, *а*, на рис. 4.11, *б* она развернута, а на рис. 4.11, *в* показано внутреннее устройство сливающих сетей, где указаны глубины проводов и значения на них для одного из возможных вариантов входов.

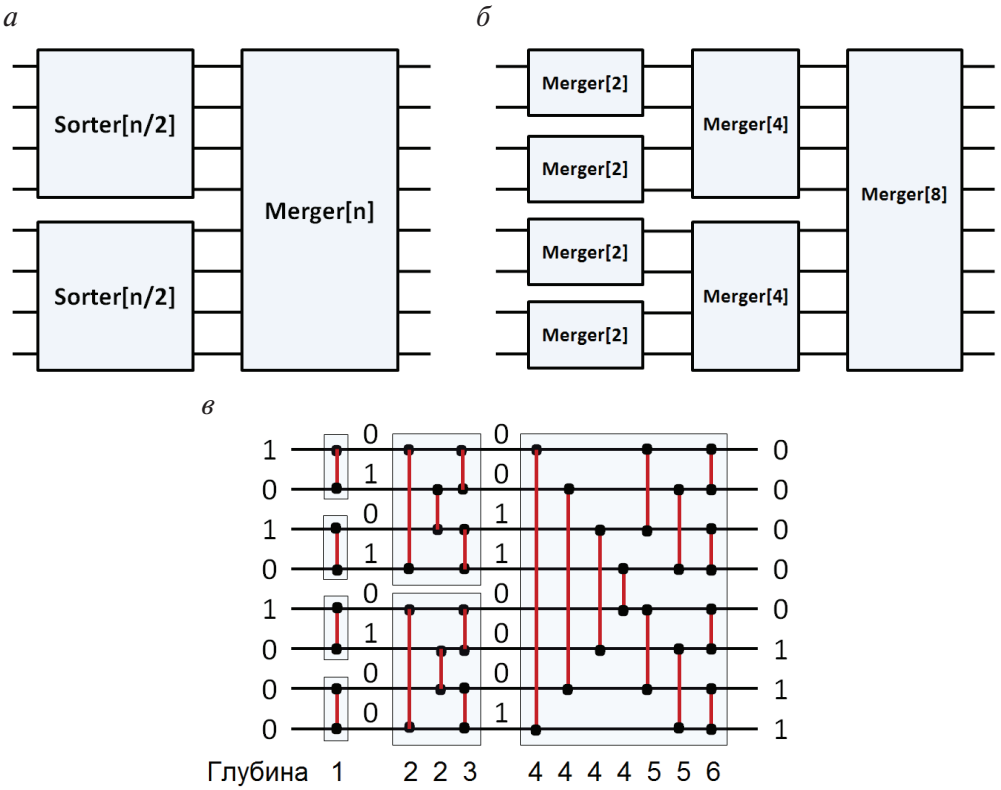


Рис. 4.11. Обобщенная структура сети `Sorter(n)`, построенная из сливающих сетей — *а*, ее рекурсивное описание — *б* и развернутый вариант при `n=8` — *в*

Таким образом, сеть `Sorter(n)` состоит из $\log_2(n)$ каскадов. Первый из них содержит $n/2$ копий сети `Merger(2)`, каждая из которых сливает две одноэлементные последовательности. Второй уровень состоит из $n/4$ копий сети `Merger(4)`, каждая из которых сливает две полученные на первом этапе двухэлементные последовательности, и т. д. На k-м уровне (при $k = 1,2,...,\log_2(n)$) имеется $n / 2^k$ копий сети `Merger(2^k)`, каждый из которых сливает две упорядоченные последовательности длиной 2^{k-1}. Индуктивное построение гарантирует, что сеть правильно

упорядочивает последовательности нулей и единиц. Из правила нуля и единицы следует правильность работы сети на произвольных числовых последовательностях.

Глубина $D(n)$ сети `Sorter(n)` равна сумме глубин сетей `Sorter(n/2)` и `Merger(n)` (две копии сети `Sorter(n/2)` работают параллельно). Глубина сети `Merger(n)` равна $\log_2(n)$, так что рекуррентная формула для $D(n)$ такова:

$$D(n) = \begin{cases} 0, & \text{если } n = 1, \\ D\left(\frac{n}{2}\right) + \log_2(n), & \text{если } n = 2k \text{ и } k \geq 1. \end{cases} \quad (4.3)$$

Из нее следует, что $D(n) = O(\log_2(n)^2)$.

Таким образом, можно сказать, что параллельный алгоритм сортировки может упорядочить n чисел за время $O(\log_2(n)^2)$, используя сортирующие сети как модель параллельных вычислений.

Вопросы и задания к главе 4

1. Каковы будут значения на проводах сети на рис. 4.2, если на ее вход подаются числа (9, 6, 5, 2)?
2. Пусть n — степень двойки. Постройте сеть компараторов с n входами и n выходами глубиной $\log_2(n)$, которая находит наименьший и наибольший из входов (и выдает их на верхний и нижний провода соответственно).
3. Профессор утверждает, что в произвольном месте сортирующей сети можно добавить компаратор и сеть останется сортирующей. Покажите, что это не так, добавив компаратор в сеть рис. 4.2.
4. Докажите, что любая сортирующая сеть с n входами имеет глубину не меньше, чем $\log_2(n)$.
5. Докажите, что любая сортирующая сеть содержит не менее $O(n\log_2(n))$ компараторов. Примечание: Наилучшие известные [32] значения при $n \to \infty$ определяются асимптотической формулой $\frac{1}{4}n(\log_2(n))^2 - \alpha\log_2(n)$ компараторов, где $\alpha = \frac{1}{4} + \frac{1}{8}\sum_{k\geq 0} 2^{-2^k-k} \approx 0{,}3569$. Например, к тому времени была найдена самая эффективная сеть для 16 входов глубиной 10 и с 60 компараторами.

7. Сеть компараторов с n входами и c компараторами представима в виде упорядоченного списка, содержащего c пар натуральных чисел от 1 до n. Паре (i, j) соответствует компаратор, соединяющий i-ю и j-ю прямые. Список перечисляет компараторы слева направо. Используя это представление, придумайте последовательный алгоритм, который находит глубину сети за время $O(n + c)$.
8. В сортирующей сети некоторые элементы являются перевернутыми компараторами, выдающими на верхний выход максимальный из входов, а на нижний — минимальный. Как преобразовать ее в сортирующую сеть из такого же числа обычных компараторов? Начните решение задачи с одного перевернутого компаратора.
9. Ко всем членам упорядоченной последовательности применили монотонно возрастающую функцию f. Останется ли последовательность упорядоченной?
10. Докажите, что сеть компараторов с n входами правильно упорядочивает последовательность $n, n-1, \ldots, 1$ тогда и только тогда, когда она правильно упорядочивает все n последовательностей нулей и единиц: (1,0,..,0), (1,1,..,0), ..., (1,1,..,1,0).
11. Используя правило нуля и единицы, докажите, что сеть компараторов на рис. 4.12 — сортирующая.

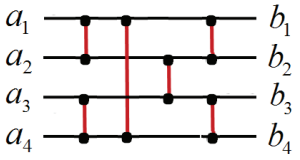


Рис. 4.12. Сортирующая сеть

12. Докажите, что для любого $i = 1, 2, \ldots, n-1$ сортирующая сеть обязана содержать хотя бы один компаратор, соединяющий прямые i и i+1.
13. Сколько всего существует различных битонических последовательностей нулей и единиц длиной n?
14. Постройте битонический сортировщик с n входами глубиной $O(\log(n))$ для случая, когда n не является степенью двойки.
15. Полуочиститель получает на вход произвольную битоническую последовательность. Докажите, что верхняя и нижняя половины выходной последовательности — битонические, а любой эле-

мент верхней половины меньше любого элемента нижней (или равен ему).

16. Даны две последовательности нулей и единиц, причем любой элемент первой меньше любого элемента второй или равен ему. Докажите, что хотя бы одна из последовательностей — чистая, т. е. нулевая.
17. Докажите правило нуля и единицы для битонических последовательностей: если сеть компараторов упорядочивает все битонические последовательности нулей и единиц, то эта сеть упорядочивает любую битоническую последовательность.
18. Докажите следующий вариант правила нуля и единицы для сливающих сетей: если сеть компараторов правильно сливает любые две монотонные последовательности нулей и единиц, то она правильно сливает любые монотонные числовые последовательности.
19. Сколько тестовых последовательностей нулей и единиц необходимо, чтобы проверить, что сеть компараторов действительно является сливающей?
20. Докажите, что любая сеть компараторов, умеющая правильно сливать один элемент a_1 с упорядоченной последовательностью $a_2, a_3, \ldots, a_n$ длиной $n-1$ в упорядоченную последовательность длиной n, имеет глубину как минимум $\log_2(n)$.

5. Нейронные сети

Нейронные сети — это раздел искусственного интеллекта, в котором для обработки сигналов используются явления, аналогичные происходящим в нейронах живых существ. Важнейшая особенность нейронной сети, свидетельствующая о ее широких возможностях и огромном потенциале, состоит в параллельной обработке информации всеми звеньями. При громадном количестве межнейронных связей это позволяет значительно ускорить процесс обработки информации. Во многих случаях становится возможным преобразование сигналов в реальном времени. Кроме того, при большом числе межнейронных соединений сеть приобретает устойчивость к ошибкам, возникающим на некоторых линиях. Функции поврежденных связей берут на себя исправные линии, в результате чего деятельность нейронной сети не претерпевает существенных возмущений.

Другое не менее важное свойство — способность к обучению и обобщению накопленных знаний в системе межнейронных связей. Натренированная на ограниченном множестве данных нейронная сеть способна обобщать полученную информацию и показывать хорошие результаты на данных, не использовавшихся ранее в процессе обучения (рис. 5.1).

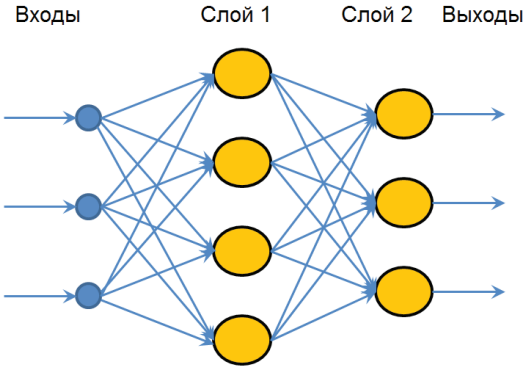


Рис. 5.1. Обобщенная структура нейронной сети

Поэтому наличие перечисленных свойств и повсеместное применение компьютерной техники вызвали в последние годы взрывной рост интереса к нейронным сетям и существенный прогресс в их исследовании. Создана база для выработки новых технологических решений, касающихся восприятия, искусственного распознания и обобщения видеоинформации, управления сложными системами, обработки речевых сигналов и т. п. [34, 35, 36].

Любая нейронная сеть используется в качестве самостоятельной системы представления знаний, которая в практических приложениях выступает, как правило, в качестве одного из компонентов системы управления либо модуля принятия решений, передающих результирующий сигнал на другие элементы, не связанные непосредственно с искусственной сетью. При этом выполняемые этой сетью (как структурным элементом) функции можно распределить на несколько основных групп: аппроксимация и интерполяция; распознавания и классификации образов; сжатия данных; прогнозирования; идентификации; управления; ассоциации.

Благодаря своим адаптационным свойствам в каждом из названных приложений нейронная сеть играет роль универсального аппроксиматора функции от нескольких переменных

$$y = f(x),$$

где x — это входной вектор, а y — реализация векторной функции нескольких переменных. Постановки значительного количества задач моделирования, идентификации и обработки сигналов могут быть сведены к подобному аппроксимационному представлению [37].

Большое число возможных приложений нейронных сетей в различных областях науки и техники делает их актуальным объектом для исследования. Поэтому в рамках настоящей главы проводится краткий обзор литературы, различных нейросетевых архитектур и формирование методического материала по использованию нейронных сетей, который может быть полезен для студентов специальностей, связанных с информационными технологиями.

5.1. Основные математические модели нейронов

Из сказанного выше следует, что каждый нейрон можно считать своеобразным процессором: он суммирует с соответствующими весами сигналы, приходящие от других нейронов, выполняет нелинейную (например, пороговую) решающую функцию и передает результирующее значение связанным с ним нейронам. В соответствии с действующим правилом «все или ничего» в простейших моделях нейронов выходной сигнал принимает двоичные значения 0 или 1. Значение 1 соответствует превышению порога возбуждения нейрона, а значение 0 — возбуждению ниже порогового уровня.

5.1.1. Модель МакКаллока-Питса

В соответствии с принципами функционирования биологических нейронов созданы различные математические модели, в которых с большей или меньшей степенью реализуются свойства природной нервной клетки [38]. Обобщенная схема, составляющая основу большинства таких моделей, восходит к представленной на рис. 5.2 модели МакКаллока-Питса, которая была одной из первых моделей нейрона [38]. В ней нейрон считается бинарным элементом, содержащим сумматор взвешенных входных сигналов и нелинейный блок обработки выходного сигнала сумматора. Структурная схема этой модели представлена на рис. 5.2.

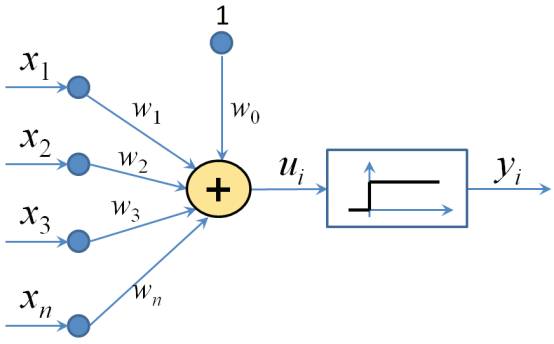


Рис. 5.2. Модель нейрона МакКаллока-Питса

Входные сигналы x_j, $j = 1, ..., N$ поступают на весовые умножители соответствующих весов w_{ij} (сигнал в направлении от входа суммируется с учетом i к узлу j) в сумматоре, после чего результат сравнивается с пороговым значением w_{i0}. Выходной сигнал нейрона y_i определяется при этом зависимостью

$$y_i = f\left(\sum_{j=1}^{N} w_{ij} x_j(t) + w_{i0}\right). \tag{5.1}$$

Аргументом функции выступает суммарный сигнал $u_i = \sum_{j=1}^{N} w_{ij} x_j(t) + w_{i0}$.

Функция $\varphi(u_i)$ называется *функцией активации*. В модели МакКаллока-Питса это *пороговая* функция вида

$$\varphi(u_i) = \begin{cases} 1, & u_i > 0; \\ 0, & u_i \le 0. \end{cases} \tag{5.2}$$

Коэффициенты w_{ij} представляют веса синаптических связей. Положительное значение w_{ij} соответствует возбуждающим синапсам, отрицательное значение w_{ij} — тормозящим синапсам, тогда как $w_{ij} = 0$ свидетельствует об отсутствии связи между i-м и j-м нейронами. Модель МакКаллока-Питса — это дискретная модель, в которой состояние нейрона в момент $(t + 1)$ рассчитывается по значениям его входных сигналов в предыдущий момент t. Построение дискретной модели обосновывается проявлением рефракции у биологических нейронов, приводящей к тому, что нейрон может изменять свое состояние с конечной частотой, причем длительность периодов бездействия зависит от частоты его срабатывания.

Через несколько лет Д. Хебб в процессе исследования ассоциативной памяти предложил теорию обучения (подбора весов w_{ij}) нейронов. При этом он использовал наблюдение, что веса межнейронных соединений при активации нейронов могут возрастать. В *нелинейной* модели Хебба приращение веса Δw_{ij} в процессе обучения пропорционально произведению выходных сигналов y_i и y_j нейронов, связанных весом w_{ij}:

$$w_{ij}(k+1) = w_{ij}(k) + \eta \cdot y_i(k) y_j(k), \tag{5.3}$$

где k означает номер цикла, η — коэффициент обучения. Согласно уравнению (5.3) в процессе обучения учитываются *корреляционные связи* между выходами нейронов.

Свойства нелинейной функции, особенно ее непрерывность, оказывают определяющее влияние на выбор способа обучения нейрона (подбор весовых коэффициентов). Другим важным фактором становится выбор стратегии обучения. Можно выделить два подхода: *обучение с учителем* и *обучение без учителя*.

При обучении с учителем предполагается, что помимо входных сигналов, составляющих вектор **X**, известны также и ожидаемые выходные сигналы нейрона d_i. Ключевым элементом процесса обучения с учителем является знание ожидаемых значений d_i выходного сигнала нейрона.

Если такой подход невозможен, остается выбрать стратегию обучения без учителя. Подбор весовых коэффициентов в этом случае проводится на основании либо конкуренции нейронов между собой (стратегии «Победитель получает все» или «Победитель получает больше»), либо с учетом корреляции обучающих и выходных сигналов (обучение по Хеббу).

Можно выделить три основных типа функций активации [37], графики зависимостей которых показаны на рис. 5.3.

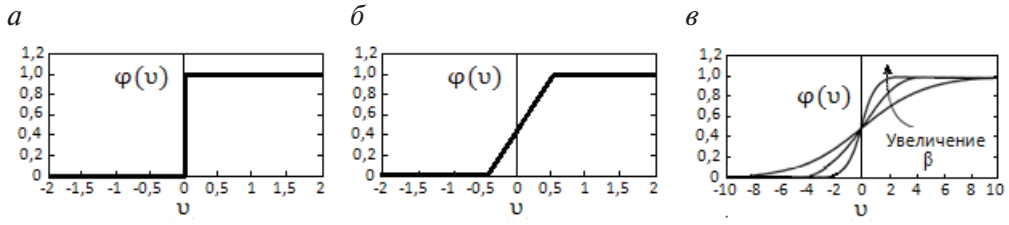


Рис. 5.3. Активационные функции: функция единичного скачка — *а*;
кусочно-линейная функция — *б*
и сигмоидальная функция для различных значений параметра β — *в*

1. *Функция единичного скачка* $\varphi(u_i)$ или пороговая функция, соответствующая на рис. 5.3, *а* формуле (5.2), где u_i — это индуцированное локальное поле нейрона.

2. *Кусочно-линейная функция*, показанная на рис. 5.3, *б*, описывается следующим выражением:

$$\varphi(u_i) = \begin{cases} 0, & u_i \le -1/2; \\ |u_i|, & -1/2 < u_i < 1/2; \\ 1, & u_i \ge 1/2, \end{cases} \qquad (5.4)$$

где коэффициент усиления в линейной области оператора предполагается равным единице. Эту функцию активации можно рассматри-

вать как аппроксимацию нелинейного усилителя. Используются также два варианта особой формы кусочно-линейной функции:

- если линейная область оператора не достигает порога насыщения, он превращается в линейный сумматор;
- если коэффициент усиления в линейной области принять бесконечно большим, то кусочно-линейная функция вырождается в пороговую функцию.

3. *Сигмоидальная функция*, график которой напоминает букву *S*, является, пожалуй, самой распространенной функцией, используемой для создания искусственных нейронных сетей. Это быстро возрастающая функция, которая поддерживает баланс между линейным и нелинейным поведением. Примером сигмоидальной функции может служить логистическая функция, задаваемая следующим выражением:

$$\varphi(u_i) = \frac{1}{1 + \exp(-\beta u_i)}, \tag{5.5}$$

где β — параметр наклона сигмоидальной функции. Изменяя этот параметр, можно построить функции с различной крутизной (см. рис. 5.3, *в*). Первый график соответствует величине параметра, равной $\beta = 1/4$. В пределе, когда параметр наклона достигает бесконечности, сигмоидальная функция вырождается в пороговую. Если пороговая функция может принимать только значения 0 и 1, то сигмоидальная функция принимает бесконечное множество значений в диапазоне от 0 до 1. При этом следует заметить, что сигмоидальная функция является дифференцируемой, в то время как пороговая такой возможностью не обладает.

Область значений функций активации, определенных формулами (5.2), (5.4) и (5.5), представляет собой отрезок от 0 до +1. Однако иногда требуется функция активации, имеющая область значений от −1 до +1. В этом случае функция активации должна быть симметричной относительно начала координат, т. е. нечетной функцией индуцированного локального поля. В частности, пороговую функцию в данном случае можно определить следующим образом:

$$\varphi(u_i) = \begin{cases} -1, & u_i \le -0; \\ 0, & u_i = 0; \\ 1, & u_i \ge 0. \end{cases} \tag{5.6}$$

Эта функция обычно называется *сигнум* или *знак*.

Для сглаживания функции знака (5.6) часто используется функция гиперболического тангенса

$$\varphi(u_i) = \tanh(\beta u_i). \tag{5.7}$$

Важным свойством функции (5.7) является ее дифференцируемость.

5.1.2. Персептрон

Простой персептрон — это обычная модель МакКаллока-Питса с соответствующей стратегией обучения. Структурная схема и обозначения элементов j-го персептрона представлены на рис. 5.4. Весовые коэффициенты входов сумматора, на которые поступают входные сигналы x_i, обозначаются w_{ij}, а пороговое значение, поступающее с так называемого поляризатора, обозначается w_{i0}. Нелинейная функция активации персептрона представляет собой дискретную функцию ступенчатого типа (5.2), вследствие чего выходной сигнал нейрона может принимать только два значения 0 или 1:

$$u_i = \sum_{j=0}^{N} w_{ij} x_j(t), \tag{5.8}$$

где u_i обозначен выходной сигнал сумматора.

В приведенной формуле подразумевается, что имеющий длину N вектор $\mathbf{x} = [x_0, x_1, x_2, ..., x_N]$ дополнен нулевым членом $x_0 = 1$, формирующим сигнал поляризации. Обучение персептрона требует наличия учителя и состоит в таком подборе весов w_{ij}, чтобы выходной сигнал y_i был наиболее близок к заданному значению d_i. Это обучение гетероассоциативного типа, при котором каждой обучающей выборке, представляемой вектором $\mathbf{x}$, априори поставлено в соответствие ожидаемое значение d_i, на выходе i-го нейрона.

Наиболее популярный метод обучения персептрона состоит в применении *правила персептрона*, в соответствии с которым подбор весов осуществляется по следующему алгоритму.

1. При первоначально выбранных (как правило, случайным образом) значениях весов w_{ij} на вход нейрона подается обучающий вектор $\mathbf{x}$ и рассчитывается значение выходного сигнала y_i. По результатам сравнения фактически полученного значения y_i с заданным значением d_i уточняются значения весов.

2. Если значение y_i совпадает с ожидаемым значением d_i, то весовые коэффициенты w_{ij} не изменяются.

3. Если $y_i = 0$, а соответствующее заданное значение $d_i = 1$, то значения весов уточняются в соответствии с формулой $w_{ij}(t+1) = w_{ij}(t) + x_j$, где t обозначает номер предыдущего цикла, а $(t+1)$ — номер текущего цикла.

4. Если $y_i = 1$, а соответствующее заданное значение $d_i = 0$, то значения весов уточняются в соответствии с формулой $w_{ij}(t+1) = w_{ij}(t) - x_j$.

По завершении уточнения весовых коэффициентов предъявляются персептрону очередной обучающий вектор **x** и связанное с ним ожидаемое значение d_i, и значения весов уточняются заново. Этот процесс многократно повторяется на всех обучающих выборках, пока не будут минимизированы различия между всеми значениями y_i и соответствующими им ожидаемыми значениями d_i [38].

5.1.3. Сигмоидальный нейрон

Нейрон сигмоидального типа имеет структуру, подобную модели МакКаллока-Питса (см. рис. 5.2), с той разницей, что функция активации является непрерывной и может быть выражена в виде сигмоидальной униполярной и/или биполярной функций. Униполярная функция, как правило, представляется формулой (5.5), тогда как биполярная функция задается в виде функции гиперболического тангенса (см. выражение (5.7)). В этих формулах параметр β подбирается пользователем. Его значение влияет на функцию активации.

Сигмоидальный нейрон, как правило, обучается с учителем по принципу минимизации целевой функции, которая для единичного обучающего набора <**x**, **d**> некоторого i-го нейрона определяется по формуле:

$$E = \frac{1}{2}(y_i - d_i)^2, \tag{5.9}$$

где $y_i = \varphi(u_i) = \varphi\left(\sum_{j=0}^{N} w_{ij} x_j\right)$.

Применение непрерывной функции активации позволяет использовать при обучении градиентные методы. Проще всего реализовать метод *наискорейшего спуска*, в соответствии с которым уточнение вектора весов $\mathbf{w} = [w_{i0}, w_{i1}, w_{i2}, ..., w_{iN}]^T$ проводится в направлении отрицательного градиента целевой функции:

$$\nabla E = \frac{dE}{dw_{ij}} = e_i x_j \frac{df(u_i)}{du_i}, \tag{5.10}$$

где $e_i = (y_i - d_i)$ означает разницу между фактическим и ожидаемым значением выходного сигнала нейрона [38].

5.1.4. Нейрон типа «адалайн»

Модель нейрона типа «адалайн» (англ.: *Adaptive Linear Neuron* — адаптивный линейный нейрон) была предложена Б. Уидроу [36]. Ее структурная схема, демонстрирующая адаптивный способ подбора весовых коэффициентов, изображена на рис. 5.4. По методу весового суммирования сигналов нейрон типа «адалайн» аналогичен представленным ранее моделям нейронов. Функция активации имеет тип «знак» (см. формулу (5.6)).

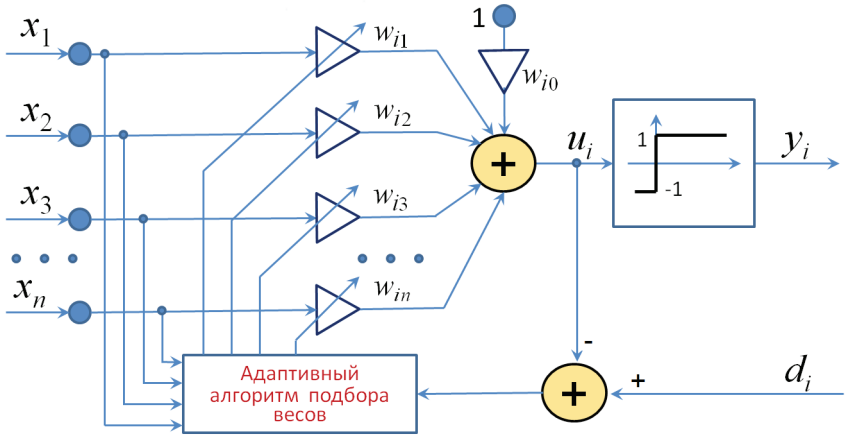


Рис. 5.4. Структурная схема нейрона типа «адалайн»

Адаптивный подбор весовых коэффициентов осуществляется в процессе минимизации квадратичной ошибки, определяемой как

$$E(\mathbf{w}) = \frac{1}{2} e_i^2 = \frac{1}{2}\left[d_i - \sum_{j=0}^{N} w_{ij} x_j \right]^2. \tag{5.11}$$

Следует обратить внимание на то, что, несмотря на нелинейный характер модели, в целевой функции присутствуют только линейные

члены, представляющие собой сумму взвешенных входных сигналов. В связи с выполнением условия непрерывности целевой функции стало возможным применение алгоритма градиентного обучения. Как и в ситуации с сигмоидальным нейроном, в алгоритме Уидроу для минимизации целевой функции применяется метод наискорейшего спуска [38].

5.1.5. Нейроны типа WTA

Нейроны типа WTA (англ.: *Winner Takes All* — победитель получает все) имеют входной модуль в виде стандартного сумматора, рассчитывающего сумму входных сигналов с соответствующими весами w_{ij}. Выходной сигнал i-го сумматора определяется согласно формуле $u_i = \sum_{j=0}^{N} w_{ij} x_j$.

Группа конкурирующих между собой нейронов (рис. 5.5) получает одни и те же входные сигналы x_j. В зависимости от фактических значений весовых коэффициентов суммарные сигналы u_i отдельных нейронов могут различаться. По результатам сравнения этих сигналов победителем признается нейрон, значение u_i, у которого оказалось наибольшим. Нейрон-победитель переходит на своем выходе в состояние 1, а остальные (проигравшие) нейроны переходят в состояние 0.

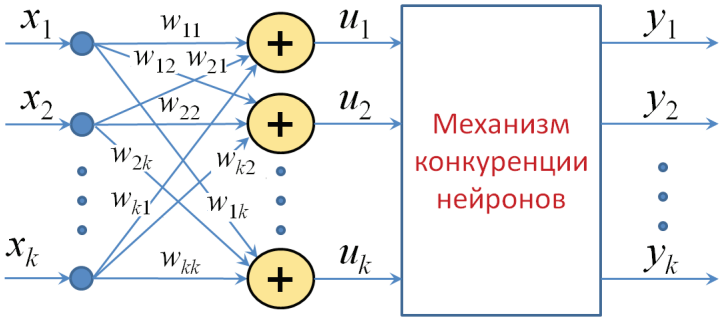


Рис. 5.5. Схема соединения нейронов типа WTA

Для обучения нейронов типа WTA не требуется учитель, оно протекает с использованием нормализованных входных векторов **x**. На начальном этапе случайным образом выбираются весовые коэффици-

енты каждого нейрона, нормализуемые относительно 1. После подачи первого вектора **x** определяется победитель этапа. Победивший в этом соревновании нейрон переходит в состояние 1, что позволяет ему провести уточнение весов его входных линий w_{ij}. Проигравшие нейроны формируют на своих выходах состояние 0, что блокирует процесс уточнения их весовых коэффициентов [38].

5.1.6. Стохастическая модель нейрона

В отличие от всех детерминированных моделей, определенных ранее, в стохастической модели выходное состояние нейрона зависит не только от взвешенной суммы входных сигналов, но и от некоторой случайной переменной, значения которой выбираются при каждой реализации из интервала (0,1).

В стохастической модели нейрона выходной сигнал y_i принимает значения ± 1 с вероятностью

$$\Pr(y_i = \pm 1) = 1 / \left[1 + \exp(\mp 2\beta u_i)\right], \tag{5.12}$$

где u_i обозначена взвешенная сумма входных сигналов i-го нейрона, а β — положительная константа, чаще всего равная 1. Процесс обучения нейрона в стохастической модели состоит из следующих этапов [38].

1. Расчет взвешенной суммы $u_i = \sum_{j=0}^{N} w_{ij} x_j$ для каждого i-го нейрона сети.

2. Расчет вероятности того, что y_i принимает значение ± 1 в соответствии с формулой (5.12).

3. Генерация значения случайной переменной $R \in (0,1)$ и формирование выходного сигнала $y_i = \pm 1$, если $R < \Pr(y_i = \pm 1)$ или $y_i = \mp 1$ в противном случае.

4. Определенный таким образом процесс осуществляется на случайно выбранной группе нейронов, вследствие чего их состояние модифицируется в соответствии с предложенным правилом.

5. После фиксации состояния отобранных нейронов их весовые коэффициенты модифицируются по применяемому правилу уточнения весов. Например, при обучении с учителем по правилу Уидроу-Хоффа адаптация весов проводится по формуле

$$\Delta w_{ij} = \eta(y_i - d_i). \tag{5.13}$$

5.1.7. Сравнение моделей абстрактных и биологических нейронов

Биологический нейрон — сложная система, математическая модель которого до сих пор полностью не построена. Введено множество моделей, различающихся вычислительной сложностью и сходством с реальным нейроном, но все они имеют ряд ограничений по сравнению с биологическими нейронами.

Вычисления выхода абстрактных нейронов предполагаются мгновенными, не вносящими задержки. Непосредственно моделировать динамические системы, имеющие «внутреннее состояние», с помощью таких нейронов нельзя. В модели отсутствуют нервные импульсы, нет модуляции уровня сигнала плотностью импульсов, как в нервной системе, не появляются эффекты синхронизации, когда скопления нейронов обрабатывают информацию синхронно, под управлением периодических волн возбуждения и торможения.

В абстрактных нейронах нет четких алгоритмов для выбора функции активации, отсутствуют механизмы, регулирующие работу сети в целом, например, гормональная регуляция активности в биологических нервных сетях.

Большое внимание в формальных моделях уделено понятиям «порог» и «весовые коэффициенты». В реальном нейроне нет числового порога, он динамически меняется в зависимости от активности самого нейрона и общего состояния сети. Весовые коэффициенты биологических синапсов тоже не постоянны и обладают пластичностью и стабильностью, весовые коэффициенты настраиваются в зависимости от сигналов, проходящих через синапс. Существует большое разнообразие биологических синапсов. Они встречаются в различных частях клетки и выполняют различные функции. Тормозные и возбуждающие синапсы реализуются в формальной модели в виде весовых коэффициентов противоположного знака, но разнообразие синапсов этим не ограничивается. Дендро-дендритные, аксо-аксональные синапсы не реализуются в модели формального нейрона.

В абстрактных моделях не прослеживается различие между градуальными потенциалами и нервными импульсами. Любой сигнал представляется в виде одного числа.

Итак, модель формального нейрона не является биоподобной и скорее похожа на математическую абстракцию, чем на живой нейрон. Тем

удивительнее оказывается многообразие задач, решаемых с помощью таких нейронов, и универсальность получаемых алгоритмов.

5.2. Модели нейронных сетей

Существенную часть в теории нейронных сетей занимают биофизические проблемы. Для построения адекватной математической модели сети необходимо детально изучить работу биологических нервных клеток и их взаимодействующих систем с точки зрения химии, физики, теории информации и синергетики. Должны быть известны ответы на следующие основные вопросы.

1. Как работает нервная клетка — биологический нейрон? Необходимо иметь математическую модель, адекватно описывающую информационные процессы в нейроне. Какие свойства нейрона важны при моделировании, а какие — нет?

2. Как передается информация через соединения между нейронами — синапсы? Как меняется проводимость синапса в зависимости от проходящих по нему сигналов?

3. По каким законам нейроны связаны друг с другом в сеть? Откуда нервная клетка знает, с какими соседями должно быть установлено соединение?

4. Как биологические нейронные сети обучаются решать задачи? Как выбираются параметры сети, чтобы давать правильные выходные сигналы? Какой выходной сигнал считается «правильным», а какой — ошибочным?

При любом объединении объектов, имеющих различные свойства, возникают системные эффекты. Тем более они появляются при соединении нейронов как формальных, так и биологических. Отметим важнейшие из свойств биологических нейросетей.

1. Параллельность обработки информации. Каждый нейрон формирует свой выход только на основе своих входов и собственного внутреннего состояния под воздействием общих механизмов регуляции нервной системы.

2. Способность к полной обработке информации. Все известные человеку задачи решаются нейронными сетями. К этой группе свойств относятся ассоциативность (сеть может восстанавливать полный образ по его части), способность к классификации, обоб-

щению, абстрагированию и множество других. Они до конца не систематизированы.

3. Самоорганизация. В процессе работы биологические нейросети самостоятельно, под воздействием внешней среды, обучаются решению разнообразных задач. Неизвестно никаких принципиальных ограничений на сложность задач, решаемых биологическими нейронными сетями. Нервная система сама формирует алгоритмы своей деятельности, уточняя и усложняя их в течение жизни. Человек пока не сумел создать систем, обладающих самоорганизацией и самоусложнением. Это свойство нейросетей рождает множество вопросов. Ведь каждая замкнутая система в процессе развития упрощается, деградирует. Следовательно, подвод энергии к нейронной сети имеет принципиальное значение. Почему же среди всех диссипативных (рассеивающих энергию) нелинейных динамических систем только у живых существ и, в частности, биологических нейросетей проявляется способность к усложнению? Какое принципиальное условие упущено человеком в попытках создать самоусложняющиеся системы?

4. Биологические нейросети являются аналоговыми системами. Информация поступает в сеть по большому количеству каналов и кодируется по пространственному принципу: вид информации определяется номером нервного волокна, по которому она передается. Амплитуда входного воздействия кодируется плотностью нервных импульсов, передаваемых по волокну.

5. Надежность. Биологические нейросети обладают фантастической надежностью: выход из строя даже 10 % нейронов в нервной системе не прерывает ее работы. И это по сравнению с последовательными ЭВМ, основанными на принципах фон Неймана, где сбой одной ячейки памяти или одного узла в аппаратуре приводит к краху системы.

Современные искусственные нейросети по сложности и интеллекту приближаются к нервной системе таракана, но уже сейчас демонстрируют ценные свойства:

1. Обучаемость. Выбрав одну из моделей нейросетей, создав сеть и выполнив алгоритм обучения, можно обучить такую сеть решению задачи, которая ей по силам. Нет никаких гарантий, что это удастся сделать при выбранных сети, алгоритме и задаче, но если все сделано правильно, то обучение бывает успешным.

2. Способность к обобщению. После обучения сеть становится нечувствительной к малым изменениям входных сигналов (шуму

или вариациям входных образов) и дает правильный результат на выходе.

3. Способность к абстрагированию. Если предъявить сети несколько искаженных вариантов входного образа, то сеть сама может создать на выходе идеальный образ, с которым она никогда не встречалась.

5.2.1. Виды нейронных сетей

В настоящее время кроме многослойного персептрона существует множество способов задания нейроподобных структур. Все виды нейронных сетей можно условно разделить на сети прямого распространения и сети с обратными связями. Как следует из названия, в сетях первого типа сигналы от нейрона к нейрону распространяются в четко заданном направлении — от входов сети к ее выходам. В сетях второго типа выходные значения любого нейрона сети могут передаваться к его же входам. Это позволяет нейронной сети моделировать более сложные процессы, например временные, но делает выходы подобной сети нестабильными, зависящими от состояния сети на предыдущем цикле [39]. На рис. 5.6 представлены наиболее распространенные типы нейронных сетей. Разнообразие нейронных сетей увеличивается еще больше благодаря огромному количеству алгоритмов и методик обучения, а также наличию нескольких видов пороговых функций.

В учебном пособии будет рассмотрен самый распространенный класс сетей — сети прямого распространения. Это — персептрон и многослойный персептрон.

5.2.2. Многослойный персептрон

Формальные нейроны могут объединяться в сети различным образом. Самым распространенным видом сети стал *многослойный персептрон*, изображенный на рис. 5.7. Сеть состоит из произвольного количества слоев нейронов. Нейроны каждого слоя соединяются с нейронами предыдущего и последующего слоев по принципу «каждый с каждым». Первый слой (слева) называется *сенсорным* или *входным*, внутренние слои называются *скрытыми* или *ассоциативными*, последний (самый правый на рисунке, состоит из одного нейрона) —

выходным или *результативным*. Количество нейронов в слоях может быть произвольным. Обычно во всех скрытых слоях одинаковое количество нейронов.

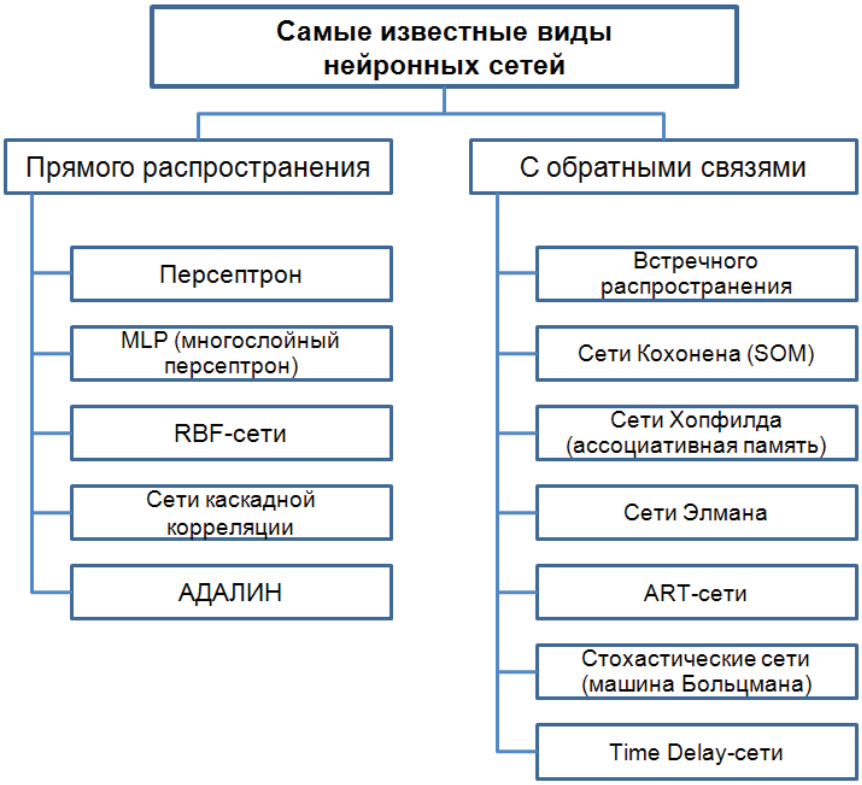


Рис. 5.6. Основные виды нейронных сетей

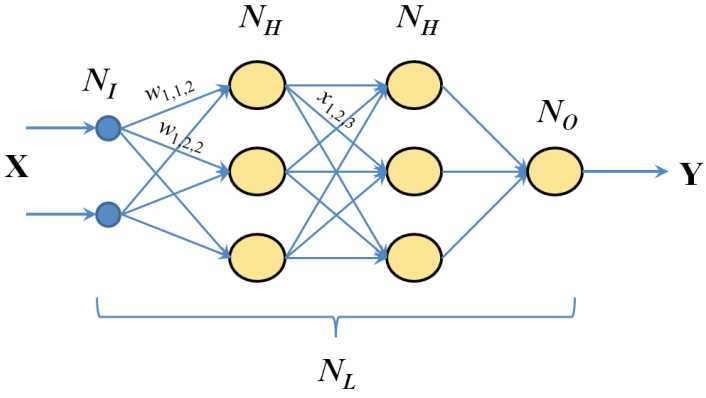


Рис. 5.7. Многослойный персептрон

Обозначим количество слоев и нейронов в слое. Входной слой имеет N_I нейронов, в каждом скрытом слое N_H нейронов и N_O выходных нейронов. Пусть, как и ранее, **x** — вектор входных сигналов сети, **y** — вектор выходных сигналов.

Существует путаница с подсчетом количества слоев в сети. Входной слой не выполняет никаких вычислений, а лишь распределяет входные сигналы, поэтому иногда его считают, иногда — нет. Обозначим через N_L полное количество слоев в сети, считая входной.

Работа многослойного персептрона (МСП) описывается формулами:

$$NET_{jl} = \sum_i w_{ijl} x_{ijl}; \tag{5.14}$$

$$OUT_{jl} = F(NET_{jl} - \theta_{jl}); \tag{5.15}$$

$$x_{ij(l+1)} = OUT_{il}, \tag{5.16}$$

где индексом i всегда будем обозначать номер входа, j — номер нейрона в слое, l — номер слоя, x_{ijl} — i-й входной сигнал j-го нейрона в слое l; w_{ijl} — весовой коэффициент i-го входа нейрона номер j в слое l; NET_{jl} — сигнал сети j-го нейрона в слое l; OUT_{jl} — выходной сигнал нейрона; θ_{jl} — пороговый уровень нейрона j в слое l. Введем также обозначения: $\mathbf{w}_{il}$ — вектор-столбец весов для всех входов нейрона j в слое l; W_l — матрица весов всех нейронов в слое l. В столбцах матрицы расположены векторы $\mathbf{w}_{il}$. Аналогично $\mathbf{x}_{jl}$ — входной вектор-столбец слоя l.

Каждый слой рассчитывает нелинейное преобразование от линейной комбинации сигналов предыдущего слоя. Отсюда видно, что линейная функция активации может применяться только для тех моделей сетей, где не требуется последовательное соединение слоев нейронов друг за другом. Для многослойных сетей функция активации должна быть нелинейной, иначе можно построить эквивалентную однослойную сеть, и многослойность оказывается ненужной. Если применена линейная функция активации, то каждый слой будет давать на выходе линейную комбинацию входов. Следующий слой даст линейную комбинацию выходов предыдущего, а это эквивалентно одной линейной комбинации с другими коэффициентами и может быть реализовано в виде одного слоя нейронов.

В многослойном персептроне нет обратных связей. Такие модели называются *сетями прямого распространения*. Они не обладают внутренним состоянием и не позволяют без дополнительных приемов моделировать развитие динамических систем.

Многослойная сеть может формировать на выходе произвольную многомерную функцию при соответствующем выборе количества слоев, диапазона изменения сигналов и параметров нейронов. Как и ряды, многослойные сети оказываются универсальным инструментом аппроксимации функций, однако наблюдается отличие работы нейронной сети

$$f(x)=F\left(\begin{array}{c} F\left(\dfrac{\sum_{i_2} w_{i_2 j_2 2}\dfrac{F\left(\sum_{i_1} w_{i_1 j_1 1}-\theta_{j_1 1}\right)}{\text{слой } 1}}{\text{слой } 2}-\theta_{j_2 2}\right) \\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot\ \cdot \\ \dfrac{\sum_{iN} w_{i_N j_N N}}{\text{слой } N}-\theta_{j_N N} \end{array}\right) \tag{5.17}$$

от разложения функции в ряд

$$f(x)=\sum\nolimits_i g_i(x). \tag{5.18}$$

За счет поочередного расчета линейных комбинаций и нелинейных преобразований достигается аппроксимация произвольной многомерной функции при соответствующем выборе параметров сети.

5.2.3. Решение информационных задач с помощью МСП

Чтобы построить МСП, необходимо выбрать его параметры. Чаще всего выбор значений весов и порогов требует *обучения*, т. е. пошаговых изменений весовых коэффициентов и пороговых уровней.

Общий алгоритм решения состоит из следующих шагов.

1. Определить, какой смысл вкладывается в компоненты входного вектора **x**. Входной вектор должен содержать формализованное условие задачи, т. е. всю информацию, необходимую для получения ответа.

2. Выбрать выходной вектор **y** таким образом, чтобы его компоненты содержали полный ответ поставленной задачи.

3. Выбрать вид нелинейности в нейронах (функцию активации $\varphi(u)$). При этом желательно учесть специфику задачи, т. к. удачный выбор $\varphi(u)$ сократит время обучения.

4. Выбрать число слоев и нейронов в слое.

5. Задать диапазон изменения входов, выходов, весов и пороговых уровней, учитывая множество значений выбранной функции активации.

6. Присвоить начальные значения весовым коэффициентам и пороговым уровням и дополнительным параметрам (например, крутизне β функции активации φ (βu), если она будет настраиваться при обучении). Начальные значения не должны быть большими, чтобы нейроны не оказались в насыщении (на горизонтальном участке функции активации), иначе обучение будет очень медленным. Начальные значения не должны быть и слишком малыми, чтобы выходы большей части нейронов не были равны нулю, иначе обучение также замедлится.

7. Провести обучение, т. е. подобрать параметры сети так, чтобы задача решалась наилучшим образом. По окончании обучения сеть готова решить задачи того типа, которым она обучена.

8. Подать на вход сети условия задачи в виде вектора **x**. Рассчитать выходной вектор **y**, который и даст формализованное решение задачи.

Многослойный персептрон может рассчитывать выходной вектор **y** для любого входного вектора **x**, т. е. давать значение некоторой векторной функции **y** = **f** (**x**). Следовательно, условием любой задачи, которая может быть поставлена персептрону, должно являться множество векторов $\{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^s\}$ с N_{in} компонентами каждый: $\mathbf{x}^s = [x_1^s, x_2^s, x_3^s, ..., x_{N_{in}}^s]^{\mathrm{T}}$. Решением задачи будет множество векторов $\{\mathbf{y}^1, \mathbf{y}^2, ..., \mathbf{y}^R\}$, каждый вектор $\mathbf{y}^r$ с N_{out} компонентами $\mathbf{y}^r = \mathbf{f}(\mathbf{x}^s)$, где $s = 1, 2, ..., S$ — номер предъявленного образа, а $r = 1, 2, ..., R$ — номер полученного после обработки образа.

Все, что способен сделать персептрон, — это сформировать отображение $X \to Y$ для $\forall x \in X$. Данное отображение полностью «извлечь» из персептрона невозможно, можно только получить отображение произвольного количества точек $\begin{pmatrix} x^1 \to y^1 \\ x^2 \to y^5 \\ x^3 \to y^R \\ \cdots \\ x^S \to y^r \end{pmatrix}$, где множество векторов $\{\mathbf{x}^1, \mathbf{x}^2, ..., \mathbf{x}^s\}$ — *формализованное условие задачи*, а множество

$\{\mathbf{y}^1, \mathbf{y}^2, ..., \mathbf{y}^R\}$ — *формализованное решение*. Задача формализации, т. е. выбор смысла, которым наделяются компоненты входного и выходного векторов, пока решается только человеком на основе практического опыта. Жестких рецептов формализации для нейронных сетей пока не создано. Рассмотрим, как выбирается смысл входных и выходных данных в наиболее распространенных случаях.

5.3. Нейронные сети в системе MATLAB

5.3.1. Приложение nnet для работы с нейронными сетями

MATLAB как язык программирования был разработан Кливом Моулером (англ. *Cleve Moler*) в конце 1970-х годов, когда он был деканом факультета компьютерных наук в Университете Нью-Мексико. Целью разработки служила задача дать студентам факультета возможность использования программных библиотек Linpack и EISPACK без необходимости изучения Фортрана. Вскоре новый язык распространился среди других университетов и был с большим интересом встречен учеными, работающими в области прикладной математики. До сих пор в Интернете можно найти версию 1982 года, написанную на Фортране, распространяемую с открытым исходным кодом. Инженер Джон Литтл (англ. *John N.* (*Jack*) *Little*) познакомился с этим языком во время визита Клива Моулера в Стэнфордский университет в 1983 году. Поняв, что новый язык обладает большим коммерческим потенциалом, он объединился с Кливом Моулером и Стивом Бангертом (англ. *Steve Bangert*). Совместными усилиями они переписали MATLAB на C и основали в 1984 году компанию The MathWorks для дальнейшего развития. Эти переписанные на языке C библиотеки долгое время были известны под именем JACKPAC. Первоначально MATLAB предназначался для проектирования систем управления (основная специальность Джона Литтла), но быстро завоевал популярность во многих других научных и инженерных областях. Он также широко использовался и в образовании, в частности, для преподавания линейной алгебры и численных методов.

В составе пакета MATLAB имеется большое количество функций для построения графиков, в том числе трехмерных, визуального ана-

лиза данных и создания анимированных роликов. Встроенная среда разработки позволяет создавать графические интерфейсы пользователя с различными элементами управления, такими как кнопки, поля ввода и другими. С помощью компонента *MATLAB Compiler* эти графические интерфейсы могут быть преобразованы в самостоятельные приложения, для запуска которых на других компьютерах необходима установленная библиотека *MATLAB Component Runtime*.

Пакет MATLAB включает различные интерфейсы для получения доступа к внешним подпрограммам, написанным на других языках программирования, данным, клиентам и серверам, общающимся через технологии Component Object Model или Dynamic Data Exchange, а также периферийным устройствам, которые взаимодействуют напрямую с MATLAB. Многие из этих возможностей известны под названием MATLAB API.

В составе современных версий системы MATLAB имеется приложение `Neural Network Toolbox` с хорошо структурированным набором функций исследования нейронных сетей.

5.3.2. Инструментарий нейронных сетей

В первую очередь рассмотрим графические инструменты этого приложения (кратко обозначим его *nnet*), которые позволяют без дополнительного знания внутреннего языка системы MATLAB провести ускоренный синтез нейронной сети, ее анализ и получить приемлемое представление о процессе и результатах обработки данных синтезированной сетью [40].

К ним относятся графические интерфейсы:

`nctool` — классификатор нейронных сетей;
`nftool` — аппроксиматор нейронной сети;
`nprtool` — распознаватель образов с помощью нейронной сети;
`nntool` — набор инструментов для синтеза и анализа сетей;
`nntraintool` — обучающий инструмент нейронной сети;
`view` — графическое представление сети и результатов.

Далее в табл. 5.1 представлены названия основных наборов функций приложения *nnet,* которые можно использовать в командном окне для детального исследования нейронной сети при обработке конкретных данных.

Таблица 5.1

Функциональная структура приложения *nnet*

Английское название	Русский перевод
`Analysis functions`	Функции анализа
`Initialize network functions`	Функции инициализации сети
`Initialize layer functions`	Функции инициализации слоя
`Initialize weight, bias functions`	Функции калибровки весов и уровней
`Learning functions`	Функции обучения
`Net input functions`	Входные функции сети
`Network creation functions`	Функции создания сети
`Network transform functions`	Функции преобразования сети
`Network update functions`	Функции обновления сети
`Performance functions`	Функции исполнения обработки
`Plotting functions`	Функции представления данных
`Processing data`	Обработка данных
`Topology functions`	Функции топологии
`Training functions`	Обучение функций
`Transfer functions`	Функции передачи
`Using networks`	Пользовательские сети
`Weight functions`	Функции весов

Пример 5.1. Нейронная сеть для выделения сигнала по образцу

В примере строится нейронная сеть net, которая далее используется для выделения сигнала X по некоторому образцу T, «примерно» напоминающему сигнал X. Функция con2seq объединяет сигнал X и образец T в вектор P из двух колонок ячеек, содержащих адреса элементов сигнала и образца. Эта функция специально написана для приложения nnet, поскольку сигнал и его образец (шаблон) могут иметь различные типы и размеры элементов. Здесь имеется ограничение: размеры самих векторов, представляющих сигнал X и образец T, должны быть одинаковы.

```
% Функция time определяет временные шаги этого моделирования.
% P определяет сигнал по этим шагам времени.
% T - сигнал, полученный из P, перемещением его влево на шаг,
% умножением на 2 и добавления всего этого к самому себе.
time = 1:0.01:2.5;
X = sin(sin(time).*time*10);
P = con2seq(X);
T = con2seq(2*[0 X(1:(end-1))] + X);
```

```
% Два подготовленных сигнала показаны на графике.
plot(time,cat(2,P{:}),time,cat(2,T{:}),'--')
title('Input and Target Signals')
xlabel('Time')
legend({'Input','Target'})

% Линейная сеть должна последовательно изменять задержку,
% чтобы обучиться (за счет перемещения во времени)
% измерению времени корреляции между P и T.
% Функция newlin создает линейный слой.
% Первый аргумент [-3 3] - ожидаемый диапазон смещения.
% Второй аргумент 1 - число нейронов в слое.
% Третий аргумент [0 1] определяет первый вход без задержки
% и второй вход с задержкой на единицу.
% Последний аргумент - параметр обучения.
net = newlin([-3 3],1,[0 1],0.1);

% Функция adapt моделирует адаптивные сети.
% Для этого требуется сеть, обрабатываемый сигнал,
% целевой (образцовый) сигнал и адаптивные фильтры сигналов.
% На графике представлен результат обработки Y в синем цвете,
% исходный сигнал T - в красном и ошибки E - в зеленом.
% t=2 сеть выявила соотношения между входом и целью, поэтому
% ошибка спадает до нуля.
[net,Y,E,Pf]=adapt(net,P,T);
plot(time,cat(2,Y{:}),'b', ...
   time,cat(2,T{:}),'r', ...
   time,cat(2,E{:}),'g',[1 2.5],[0 0],'k')
legend({'Output','Target','Error'})
```

На рис. 5.8, показывающем графики трех процессов на интервале адаптации (с 1 с до 2,5 с), видно, что уже после 0,5 с «работы» нейронной сети net она выделяет входной сигнал практически без ошибок. Веса нейрона сети настроены на группы сигналов, подобные образцу `T` или самому сигналу `X`. Нейронная сеть может быть использована, например, для очищения от высокочастотных шумов сигналов типа `T`.

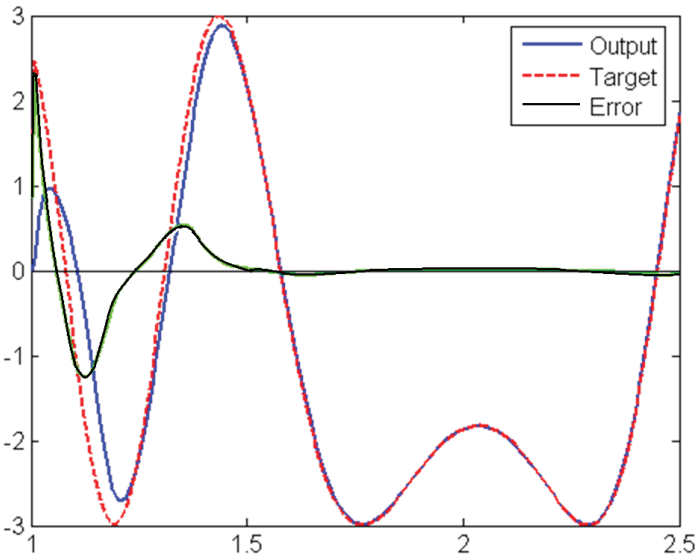


Рис. 5.8. Процесс адаптации нейросети

В составе приложений системы MATLAB имеется множество графических интерфейсов, которые широко используются для облегчения общения с пользователем, а также повышают удобство ввода и вывода различной информации. Естественно, что в приложении nnet предусмотрено большое число встроенных графических демонстрационных примеров, а также инструментальных графических программных модулей. Основной программой при работе с нейронными сетями является нейросетевой менеджер, изображенный на рис. 5.9. Он вызывается по команде `nntool`. В менеджере можно создавать, экспортировать, редактировать, удалять различные данные, используемые для работы с нейронными сетями, а также сами нейронные сети. Рассмотрим более подробно работу менеджера нейронных сетей в системе MATLAB.

Чтобы создать в нем и обучить нейронную сеть, нужно вначале импортировать данные — это входные, обучающие и выходные данные, т. е. те данные, которые будут подаваться на входы и выходы нейронной сети. При нажатии на клавишу Import в окне на рис. 5.9 появится дополнительное окно, изображенное на рис. 5.10, в котором нужно выбрать, какие данные будут входящими (Input Data), а также образцы данных (Target Data), используя которые, на выходе нейронной сети после обработки входных данных могут быть получены выходные данные (Output Data). В некоторых случаях может потребоваться

информация о предварительной настройке входных состояний и слоев или сведения об ошибках обработки данных. После импорта данных они появляются в менеджере нейронных сетей, далее можно создавать собственно нейронную сеть.

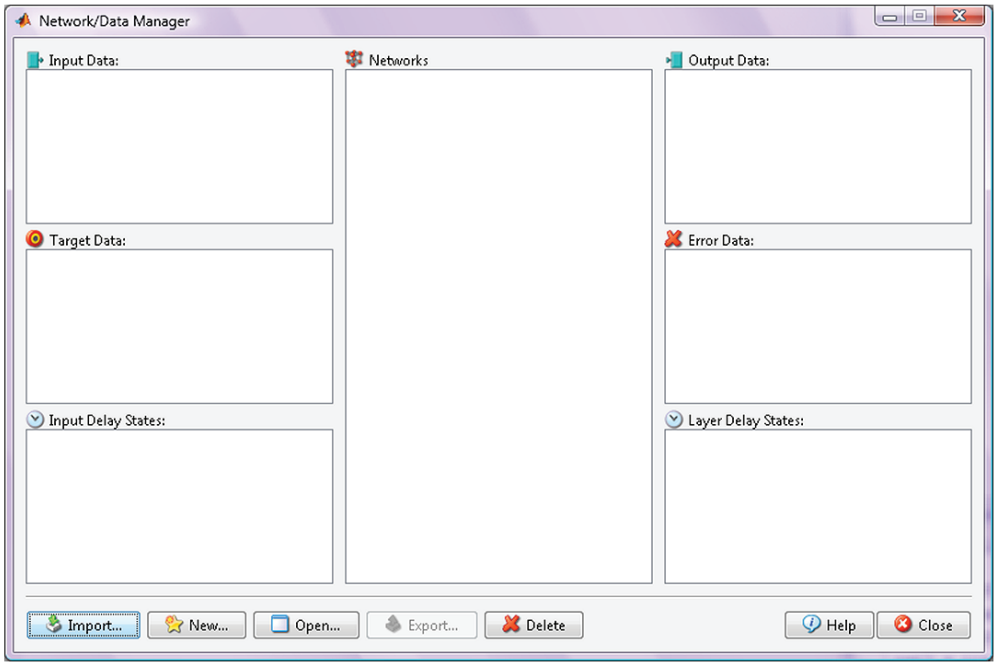


Рис. 5.9. Нейросетевой менеджер в MATLAB

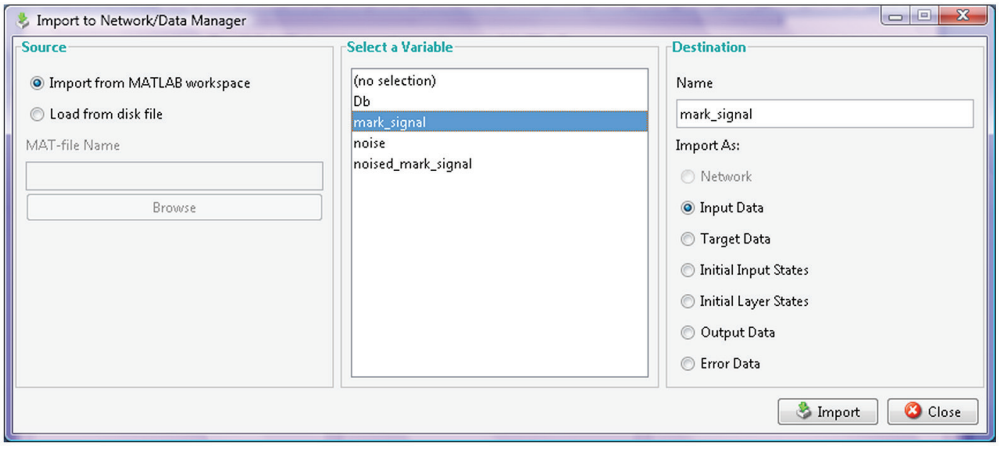


Рис. 5.10. Импорт данных в нейросетевой менеджер

В менеджере нейронных сетей, изображенном на рис. 5.9, по кнопке New можно создавать различные нейронные сети. Создадим, например, двухслойную нейронную сеть прямого распространения, при обучении которой будет использоваться алгоритм обратного распространения ошибки (рис. 5.11).

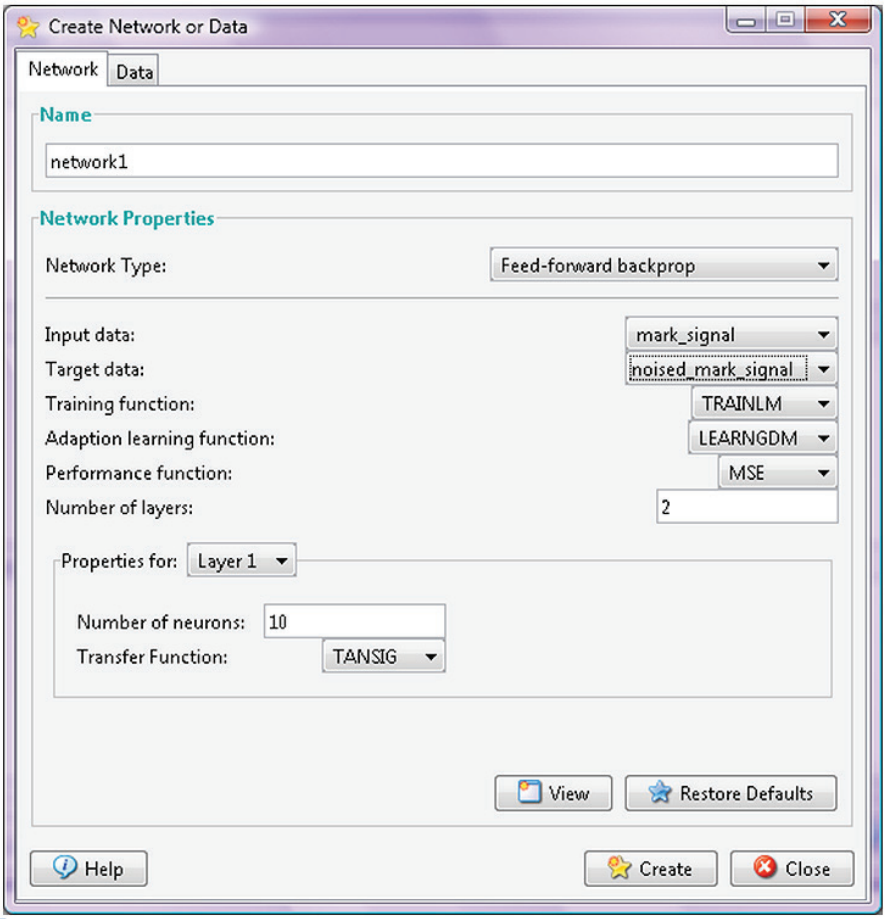


Рис. 5.11. Создание нейронной сети

Нажатие кнопки Create приводит к созданию нейронной сети network1, структура которой наглядно представлена на рабочем окне графического интерфейса Networr (см. рис. 5.12). Командная строка этого интерфейса содержит главные содержательные команды для обработки и представления введенных данных с помощью синтезированной нейронной сети.

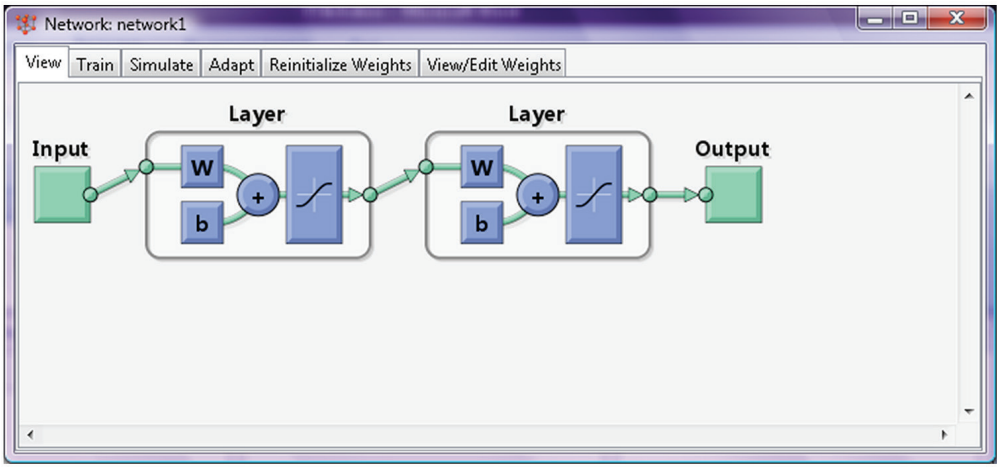


Рис. 5.12. Созданная нейронная сеть прямого распространения

На вкладке View (рис. 5.12) показан внешний вид этой нейронной сети. На других вкладках можно обучать, моделировать прохождение различных сигналов через нейронную сеть, а также посмотреть значение весов нейронов в сети и, если потребуется, вернуть их к начальному значению. Таким образом, в этом интерфейсе нейронная сеть может также редактироваться.

В приложении nnet существует демонстрационная программа, изображенная на рис. 5.13, вызываемая по команде nftool, которая демонстрирует возможности нейронных сетей в качестве обобщающей функции.

Например, в медицине можно собрать данные по болезням людей и сформировать базу данных, содержащую информацию о том, сколько человеку лет, где он живет, болезни у родственников, его жалобы и прочие антропометрические данные — это будут наши входные данные. А выходные данные — данные, которые должна выдавать нейронная сеть — рекомендации к обследованию, вероятности заболевания определенных болезней. После создания такой базы данных и обучения нейронной сети можно будет любому человеку — здоровому, или с жалобами — ввести свои данные на вход нейронной сети и получить результат, причем точность диагноза будет довольно высока. Основная особенность состоит здесь в том, что все люди уникальные, каждый отличается друг от друга, но в то же время у всех есть некие определенные черты, общие признаки, которые совпадают друг с другом.

Именно в этом плане нейронные сети очень хорошо могут обобщать данные (свойство аналоговость) и разделение весов по слоям позволяет с высокой точностью обрабатывать столь сложные и нелинейные данные.

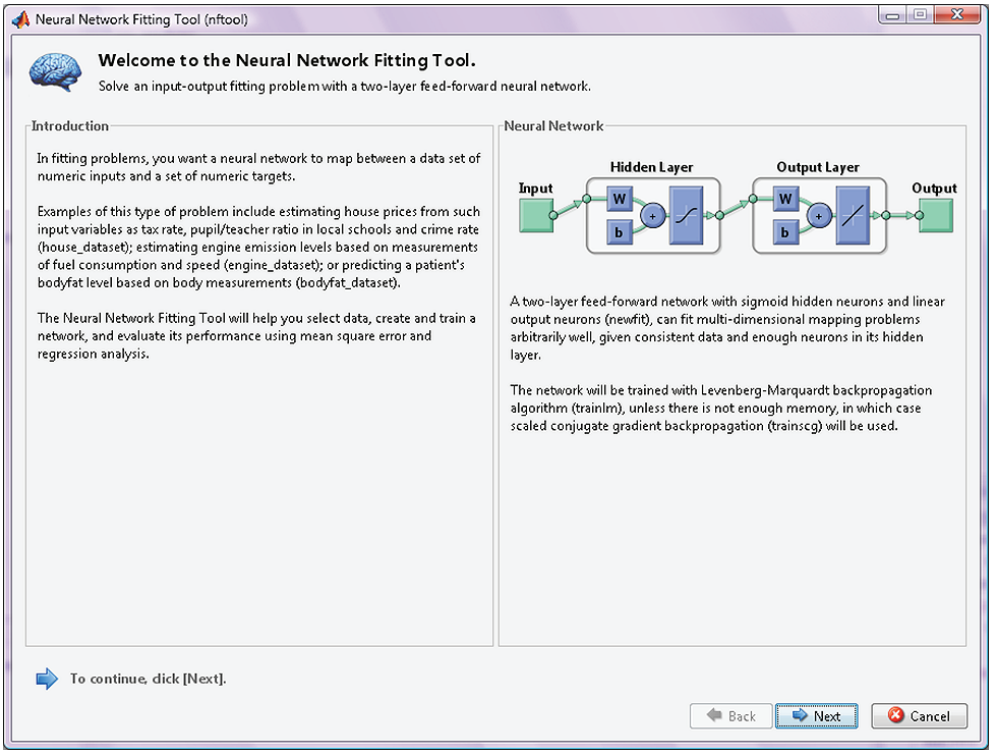


Рис. 5.13. Демонстрационная программа NN fitting tool

В программе, изображенной на рис. 5.13, нажимается кнопка Next, вызывающая на экран окно (см. рис. 5.14) для выбора демонстрационных баз данных семи различных классов. На показанном интерфейсе выбран класс Body Fit или аппроксимация параметров человеческого тела, а среди 13 возможных органов можно провести анализ свойств от глаза (Age) до запястья (Wrist circumference).

В программе используется персептрон с двумя скрытыми слоями, причем функция активации нейронов в первом слое сигмоидальная, а во втором слое находится один нейрон с линейной функцией активации. После ввода демонстрационных баз данных на экране появля-

ется графический интерфейс для обучения нейронной сети, а также основные числовые параметры процессов обработки нейронной сети. В ходе обучения можно корректировать количество нейронов в первом скрытом слое и другие параметры сети.

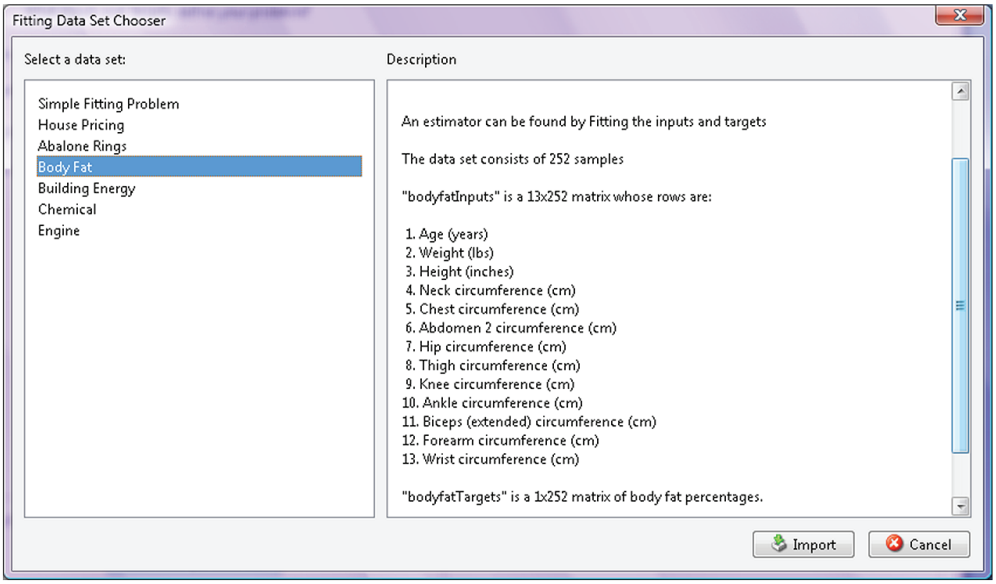


Рис. 5.14. Демонстрационные базы данных

После окончания тренировки будет выведено окно, изображенное на рис. 5.15, в котором представлены числовые характеристики результата обучения, по которым можно судить, насколько хорошо выбранная нейронная сеть справилась с задачей аппроксимации (или обобщения) текущей базы данных. В окне указаны пояснения для значений MSE и R-value.

При нажатии кнопки Next в окне на рис. 5.15 будет выведено следующее окно, изображенное на рис. 5.16, где предлагается проверить работу обученной нейронной сети с новым набором данных, если таковой имеется.

В том случае, когда результат работы с сетью устраивает, можно обученную нейронную сеть импортировать либо в рабочее пространство (Workspace), либо в Simulink, либо сохранить в формате *m*-файла для дальнейшей работы (см. рис. 5.17).

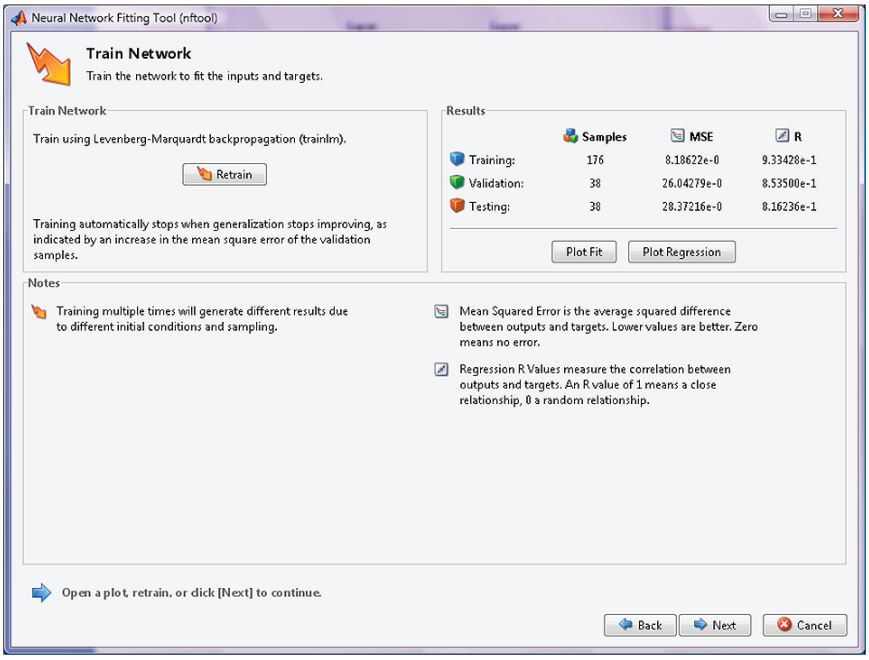


Рис. 5.15. Данные о нейронной сети после тренировки

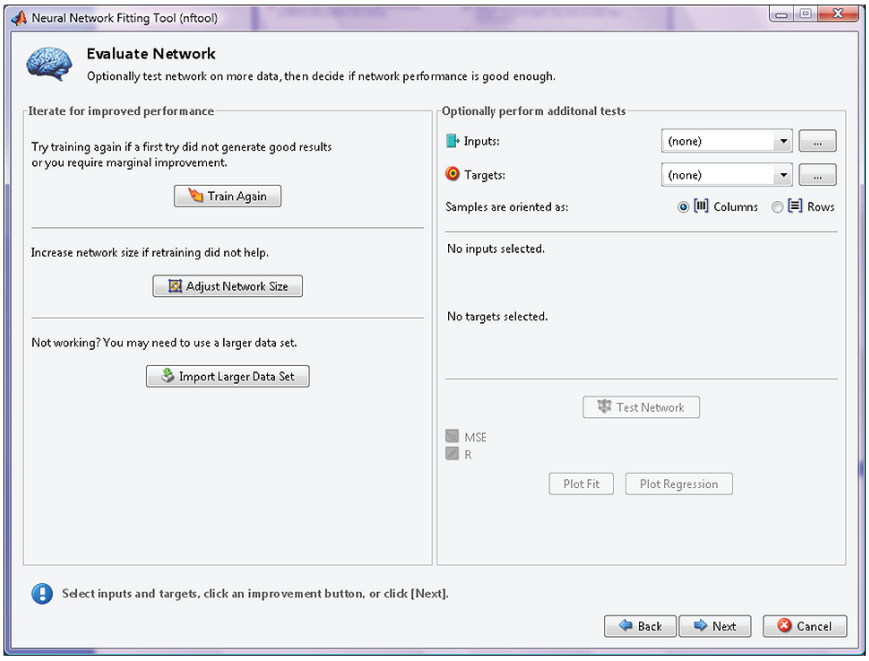


Рис. 5.16. Дополнительная проверка нейронной сети

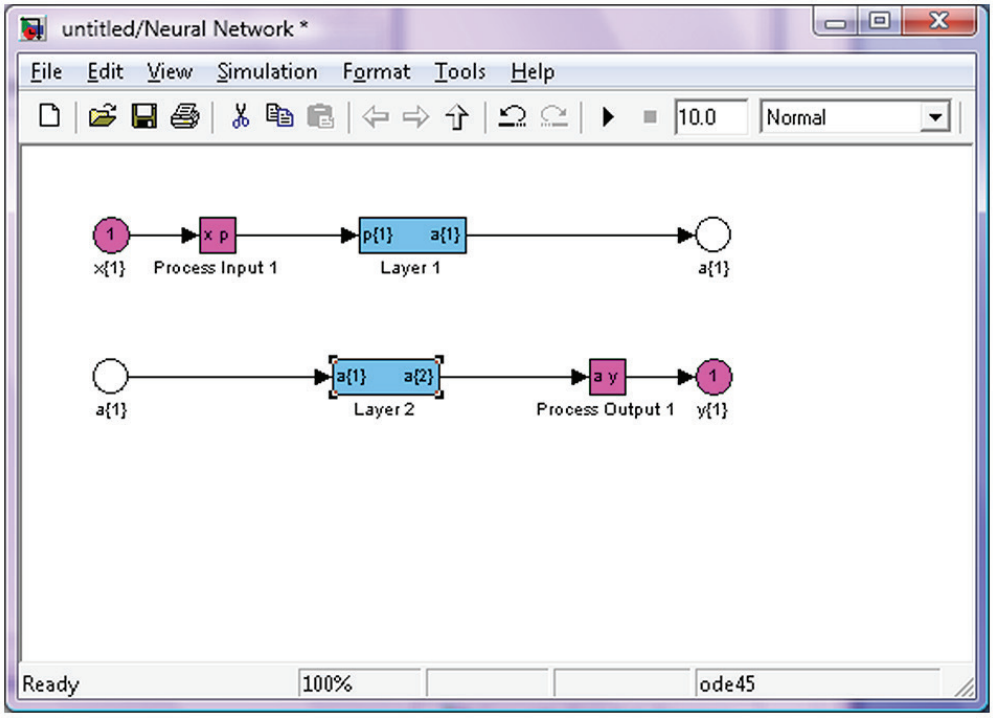


Рис. 5.17. Модель нейронной сети в Simulink

Помимо представленных здесь инструментов с графическим интерфейсом в MATLAB предусмотрены определенные функции для создания нейронной сети прямо из командной строки или через текстовый редактор *m*-файлов, как это было показано выше в примере 5.1.

5.3.3. Моделирование нейронной сети прямого распространения

В параграфе изучается работа нейронной сети прямого распространения — многослойного персептрона, изображенного на рис. 5.7.

Сеть состоит из произвольного количества слоев нейронов. Нейроны каждого слоя соединяются с нейронами предыдущего и последующего слоев по принципу «каждый с каждым». Первый слой (слева) называется *сенсорным* или *входным*, внутренние слои называются *скрытыми* или *ассоциативными*, последний (самый правый на рис. 5.7, который состоит из одного нейрона) — *выходным* или *результатив-*

ным. Количество нейронов в слоях может быть произвольным. Обычно во всех скрытых слоях одинаковое количество нейронов.

В системе MATLAB написано небольшое графическое приложение, представленное на рис. 5.18. Это графическое окно, где предложены для загрузки на выбор пользователя заранее подготовленные базы данных. Эти данные нужны для синтеза и обучения нейронной сети. В графическом окне также приведено краткое описание того, какая база данных будет загружена, т. е. какие данные будут подаваться на входы и выходы нейронной сети.

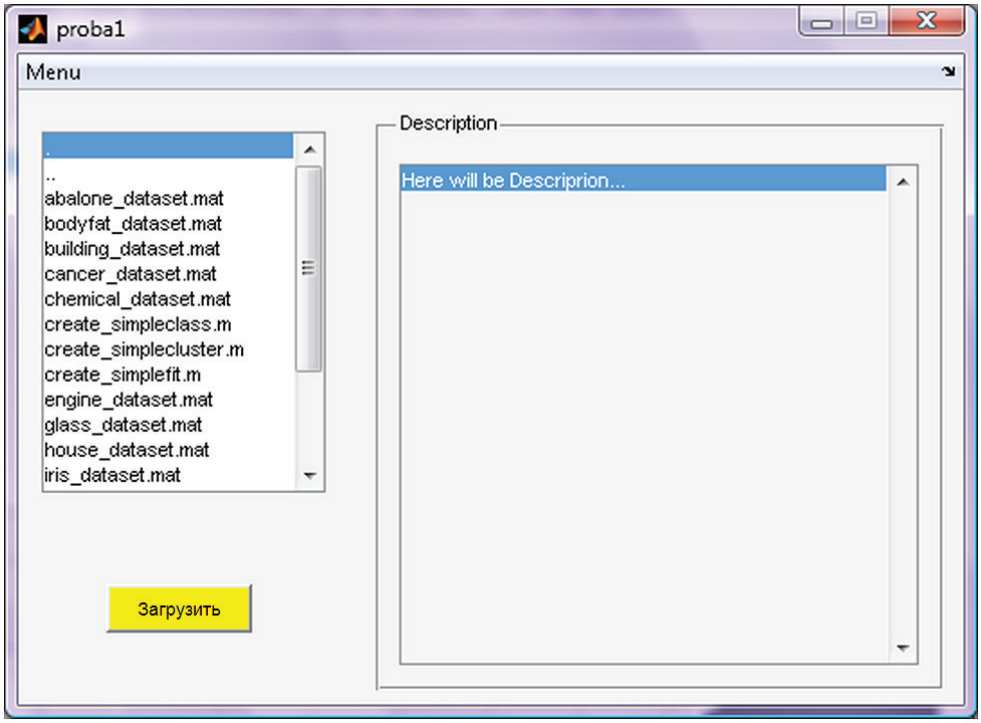


Рис. 5.18. Графический интерфейс для загрузки обучающих баз данных

С левой стороны находится колонка всех встроенных в пакет MATLAB обучающих баз данных для нейронных сетей, справа — описание выбранной базы данных. Чтобы загрузить нужную базу данных, необходимо выбрать ее в левом списке и нажать кнопку Загрузить, либо щелкнуть по выбранному пункту два раза мышью. После загрузки базы данных в рабочем пространстве появятся две перемен-

ные с именами Inputs и Targets, соответствующие входным и выходным данным для обучения нейронной сети.

Обучение такой нейронной сети будет проводиться с помощью встроенных в приложение nnet функций. Алгоритм обучения сетей прямого распространения — это алгоритм Румельхарта-Хинтона-Вильямса, т. е. алгоритм обратного распространения ошибки (теоретическое описание алгоритма приведено в [38]).

После загрузки данных нужно создать нейронную сеть и ее обучить. Для создания нейронной сети воспользуемся функцией **newff**, которая в приложении nnet системы MATLAB синтезирует сеть прямого распространения:

```
net = newff (Inputs, Targets, [2020]);
```

Согласно выходному параметру команды создается нейронная сеть с именем **net**, входные переменные **Inputs** и **Targets** — имена векторов (массивов), где содержатся входные и выходные обучающие последовательности данных. Третий входной параметр команды — двухэлементный массив **[20 20]** — показывает, сколько нейронов будет в скрытых слоях нейронной сети, т. е. в данном случае синтезирована сеть, состоящая из двух скрытых слоев по 20 нейронов в каждом.

После выполнения команды в рабочем пространстве появится переменная — объект **net**, который представлен структурой типа «Нейронная сеть». Два раза щелкнув по ней мышью, можно посмотреть содержимое полей созданной структуры нейронной сети (см. рис. 5.19).

Для обучения нейронной сети в MATLAB существует функция **train**:

```
net = train (net, Inputs, Targets);
```

В этой команде **net** — имя нейронной сети, которую будут обучать, **Inputs** и **Targets** — имена переменных, в которых находятся входные и выходные данные, с помощью которых и будет проходить обучение нейронной сети.

В окне, изображенном на рис. 5.20, показано, сколько итераций (эпох) обучений пройдено, время обучения и его производительность. После обучения можно посмотреть график обучения (plotperform), нажав кнопку Performance.

Из графиков, изображенных на рис. 5.21, видно, что минимальное значение MSE (СКО или среднеквадратичное отклонение) составило величину 39.97. Такой результат обучения нейронной сети следует считать неудовлетворительным. В этом случае нужно изменить количество нейронов в скрытых слоях или само количество слоев и повто-

рить обучение. Как было сказано выше, количество нейронов в нейронной сети для ее оптимальной работы достигается эмпирическим путем, т. е. путем многократного повторения процесса обучения при различных параметрах сети. Очевидно, что чем меньше MSE, тем лучше. Поэтому, если есть возможность, то нужно добиваться минимального значения СКО. Хорошим результатом обучения в данном случае является результат при MSE $\leqslant 10$.

Рис. 5.19. Содержание свойств (полей структуры) нейрообъекта net

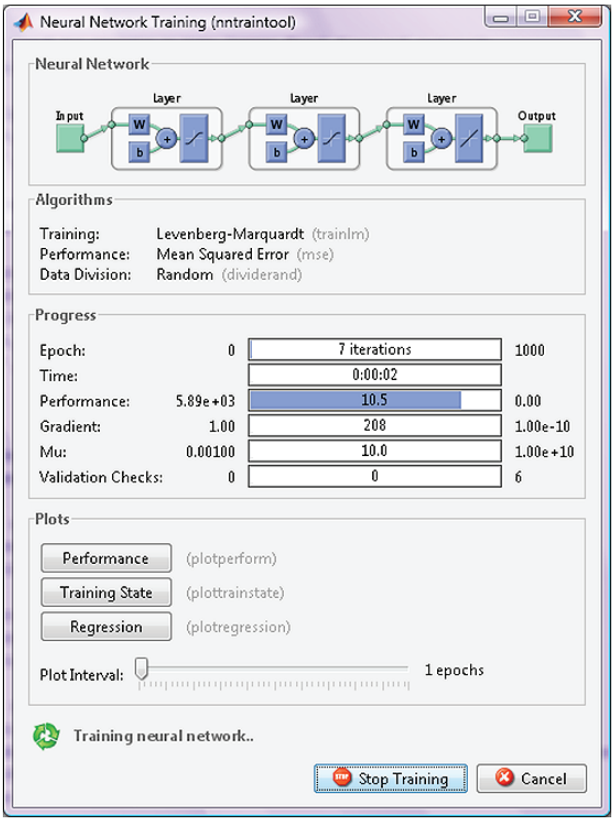


Рис. 5.20. Интерфейс для контроля процесса обучения нейронной сети

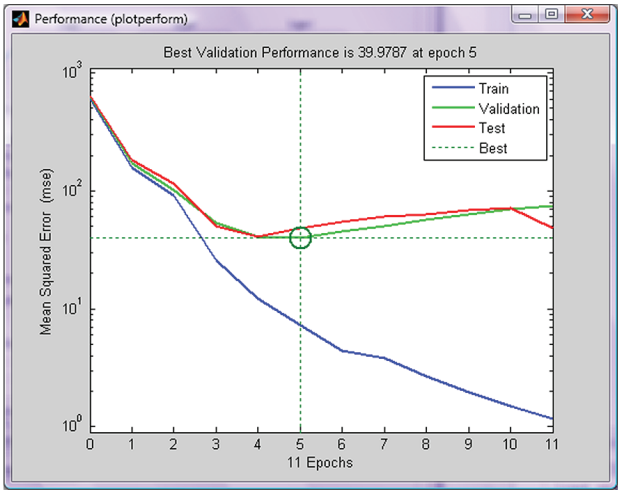


Рис. 5.21. Результаты обучения нейронной сети

При достижении удовлетворительного результата обучения рекомендуется использовать функцию **sim** (sim — сокращение от simulate — моделировать), с помощью которой моделируется прохождение сигнала через нейронную сеть:

```
Outputs = sim (net, Inputs);
```

В этой команде переменной **Outputs** присваивается последовательность данных, которая образуется при прохождении через обученную сеть net при подаче на ее вход данных **Inputs**, использованных при обучении. Тогда можно вычислить относительное СКО — это позволит оценить, насколько хорошо обучена нейронная сеть. Выполнить это можно с помощью команды

```
MSE = std (Targets-Outputs)/std (Targets);
```

в которой находится относительная ошибка обучения. Функция **std** вычисляет СКО данных, поэтому отношение СКО разности обучающей последовательности и выходной последовательности к СКО обучающей последовательности и даст искомую оценку качества обработки нейронной сети. Сеть считается хорошо обученной, если значение MSE не выше примерно 0,4.

Если требуемое условие для MSE выполнилось, можно (и нужно) смоделировать прохождение различной информации из других, не использовавшихся для обучения баз данных, чтобы показать, что нейронная сеть работает адекватно в качестве обобщающей функции. Этот процесс работы с нейронной сетью называется *валидацией*.

5.3.4. Обработка сигналов с помощью нейронных сетей

Задача настоящего пункта состоит в исследовании нейронной сети при выделении смоделированного речевого сигнала на фоне шума и помех, поэтому здесь сеть будет выступать в роли адаптивного фильтра.

Под термином фильтр обычно понимают устройство или алгоритм, используемые для извлечения полезной информации из набора зашумленных данных. Шум может возникать по многим причинам. Например, данные могут быть измерены с помехами, или возмущения могут возникнуть при передаче информационного сигнала через зашумленные линии связи. Кроме того, на полезный сигнал может быть наложен другой сигнал, поступающий из окружающей среды в виде по-

мехи. Фильтры можно применять для решения трех основных задач обработки информации.

1. *Фильтрация*, т. е. актуальное извлечение полезной информации в дискретные моменты времени из текущих данных.

2. *Сглаживание*. Эта задача отличается от фильтрации тем, что информации о полезном сигнале непосредственно во время обработки не требуется, поэтому для извлечения этой информации можно сразу использовать все данные анализируемого процесса. Поскольку при сглаживании можно использовать данные, полученные не только до некоторого момента времени, но и после него, то этот процесс в статистическом смысле является более точным, чем фильтрация.

3. *Прогнозирование*. Целью этого процесса является получение прогноза относительно состояния объекта управления в некоторый будущий момент времени $t + \Delta t$, где $\Delta t > 0$, на основе данных, полученных до момента t (включительно).

Важным для практики является применение адаптации. Например, одной из областей применения адаптивной фильтрации является очистка данных от нестабильных мешающих сигналов и шумов, перекрывающихся по спектру со спектром полезных сигналов, или когда полоса мешающих частот неизвестна, переменна и не может быть задана априори для расчета параметрических фильтров. Частотная характеристика адаптивных фильтров автоматически регулируется или модифицируется в соответствии с определенным критерием, позволяющим фильтру подстраиваться к изменениям характеристик входного сигнала. Общая структура адаптивного фильтра показана на рис. 5.22.

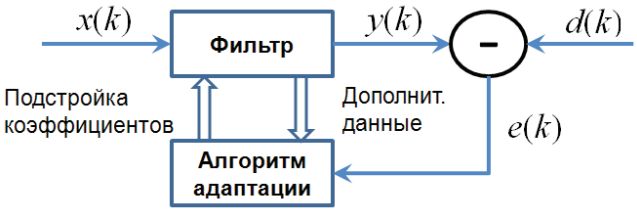


Рис. 5.22. Общая структура адаптивного фильтра

Входной сигнал $x(k)$ обрабатывается фильтром, в результате чего получается выходной сигнал $y(k)$. Этот выходной сигнал сравнивается с *образцовым* сигналом $d(k)$, разность между ними образует сигнал ошибки $e(k)$. Задача адаптивного фильтра — минимизировать ошибку

воспроизведения образцового сигнала. С этой целью блок адаптации после обработки каждого отсчета анализирует сигнал ошибки и дополнительные данные, поступающие из фильтра, используя результаты этого анализа для подстройки параметров (коэффициентов) фильтра.

Возможен и иной вариант адаптации, при котором образцовый сигнал не используется. Такой режим работы называется *слепой адаптацией* или *обучением без учителя* (*unsupervised learning*). Разумеется, в этом случае требуется некоторая информация о структуре полезного входного сигнала (например, знание типа и параметров используемой модуляции). Очевидно, что слепая адаптация является более сложной вычислительной задачей, чем адаптация с использованием образцового сигнала.

Для выполнения поставленной в начале пункта задачи нужна модель речевого сигнала. Такой моделью чаще всего выбирается марковский процесс второго порядка [25]. Смоделированный марковский процесс смешивается с шумом при отношении сигнал/шум регулируемой переменной Q. По умолчанию отношение сигнал/шум выражается в децибелах. Другими вариантами моделей помех могут быть немодулированный гармонический сигнал или модулированный по амплитуде, частоте, фазе и т. п. Количество отсчетов всего входного процесса должно быть не менее нескольких тысяч. На рис. 5.23 приведен график смоделированного речевого сигнала — марковского процесса второго порядка.

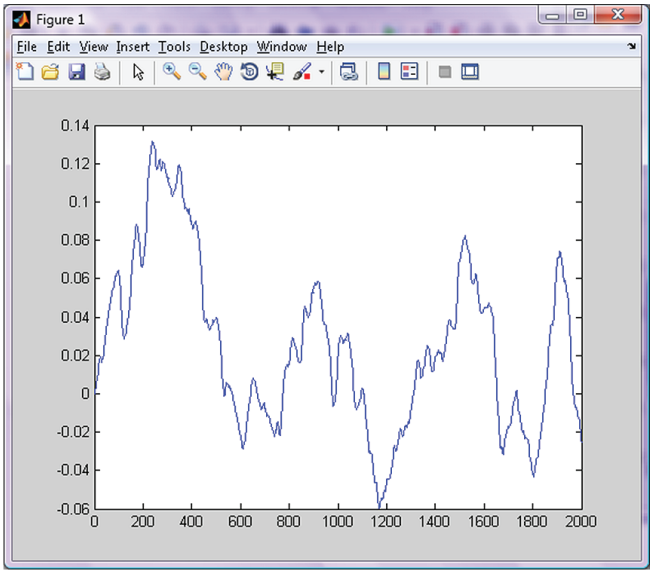


Рис. 5.23. Марковский процесс второго порядка

Результат аддитивного смешивания речевого сигнала с белым гауссовским шумом при $Q = 4$ показан на рис. 5.24.

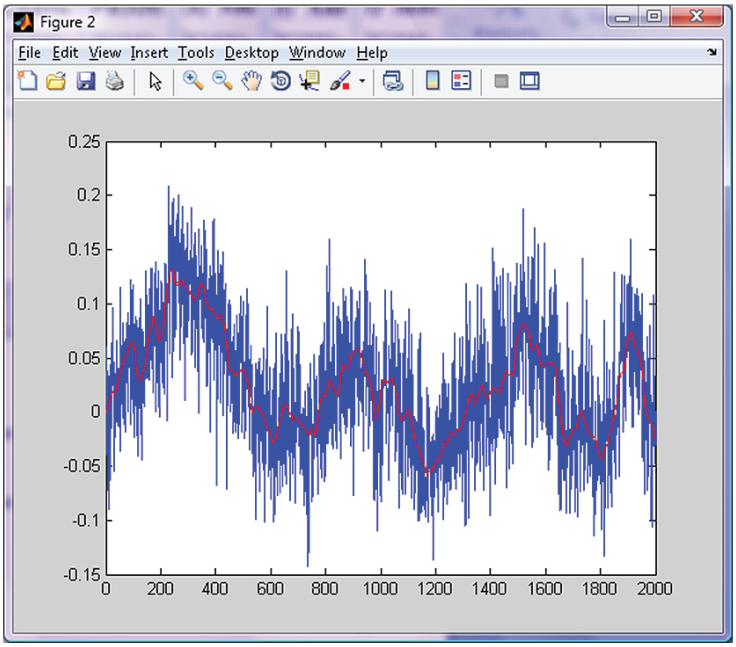


Рис. 5.24. Модель речевого сообщения на фоне белого шума

После моделирования речевого сигнала и наложения на него шума (помехи) в рабочем пространстве будут находиться две числовые последовательности. В одной из них — смоделированный речевой сигнал, в другой — этот же смоделированный сигнал, смешанный с помехой. Количество ранее заданных отсчетов равно числу элементов числовой последовательности.

Создание нейронной сети можно выполнить как с помощью нейросетевого менеджера nntool, так и с помощью текстовой команды, например,

```
net = newff (Inputs, Targets, [220 20]);
```

В любом случае после создания нейронной сети рекомендуется с ней работать в менеджере нейронных сетей, поскольку при оформлении результатов потребуется рисунок нейронной сети, используемой для подавления помехи в речевом сигнале. Менеджер позволяет наглядно отображать действия, выполняемые с нейронной сетью (см. рис. 5.25).

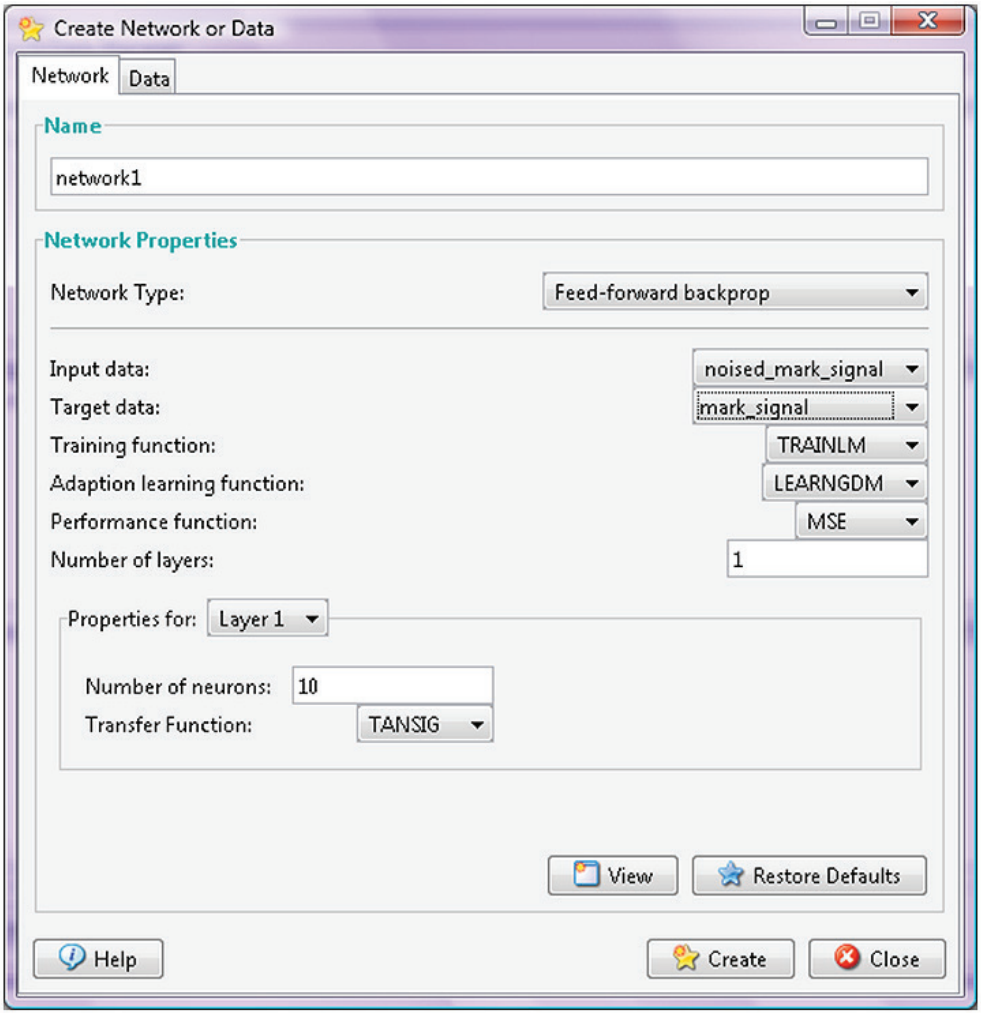


Рис. 5.25. Создание нейронной сети для выделения речевого сигнала

На рис. 5.25 показаны действия при создании нейронной сети: какие числовые последовательности подаются на вход сети, что ожидается на ее выходе. Соответственно, на вход Inputs подается смоделированный речевой сигнал с наложенной на него помехой noised_mark_signal, а с выхода Targets ожидается начальный речевой смоделированный сигнал без помех mark_signal.

После того как создана нейронная сеть и правильно поданы на входы и выходы данные, можно начинать ее обучение (см. рис. 5.26). Обучение проводится аналогично обучению, описанному в предыдущем пункте.

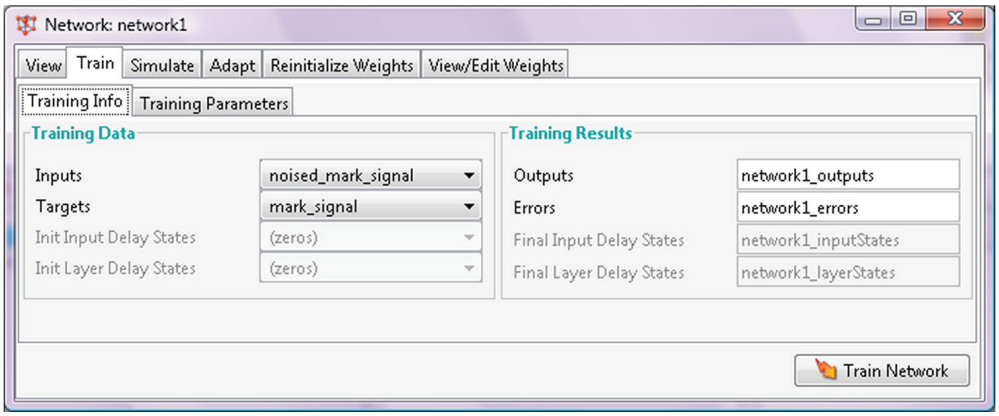


Рис. 5.26. Обучение нейронной сети

Можно начать обработку сигнала сетью с одним скрытым слоем и, если будет нужно, потом увеличить количество слоев и нейронов в сети. Чем сильнее искажается помехой смоделированный речевой сигнал и чем меньше отношение сигнал/шум, тем больше нужно нейронов в сети и тем дольше будет проходить обучение.

После того как сеть прошла обучение, следует заново смоделировать речевой сигнал и тот же сигнал с наложенным на него шумом и эти две новые числовые последовательности обработать с помощью обученной нейронной сети, т. е. выполнить процесс валидации при новых случайных данных. По ее результатам необходимо определить величину нормированной среднеквадратической ошибки. Рекомендуется произвести эти вычисления для нескольких нейронных сетей, чтобы получить некую корреляцию результатов обработки данных с помощью нейронных сетей и количеством нейронов в сети.

5.3.5. Использование нейронных сетей в распознавании образов

С задачей распознавания образов живые системы, в том числе и человек, сталкиваются постоянно с момента своего появления. В частности, информация, поступающая с органов чувств, обрабатывается мозгом, который в свою очередь сортирует информацию, обеспечивает принятие решения, а далее с помощью электрохимических импульсов передает необходимый сигнал, например, органам движения, ко-

торые реализуют необходимые действия. Если происходит изменение окружающей обстановки, то названные выше процессы повторяются. При этом каждый этап сопровождается распознаванием.

В процессе жизнедеятельности человека число принимаемых им решений конечно, но, в то же время, количество определяющих факторов может быть бесконечным. Количество возможных решений зависит от жизненного опыта. Поэтому автоматизация ряда процессов предполагает конструирование автоматических устройств, способных реагировать на множество изменяющихся характеристик внешней среды каким-то определенным количеством удовлетворительных для человека реакций. Это означает реализацию главных особенностей принципов распознавания, заложенных природой, за счет возможности реагирования на совокупность изменений.

Создание устройств, выполняющих функции распознавания различных объектов, в большинстве случаев обеспечивает возможность замены человека специализированным автоматом. Благодаря этому значительно расширяются возможности сложных систем, выполняющих различные информационные, логические, аналитические задачи. Следует отметить, что качество работ, выполняемых человеком на рабочем месте, зависит от многих факторов (квалификации, опыта, добросовестности и т. д.). В то же время исправный автомат действует однообразно и обеспечивает всегда одинаковое качество. Автоматический контроль сложных систем позволяет вести мониторинг и обеспечивать своевременное обслуживание, идентификацию помех и автоматическое применение соответствующих методов шумоподавления, позволяет повысить качество передачи информации. Также понятно, что использование автоматических систем в ряде задач может обеспечить невозможное для человека быстродействие.

Прежде чем приступить к основным методам распознавания образов, приведем несколько необходимых определений.

Распознавание образов (объектов, сигналов, ситуаций, явлений или процессов) — задача идентификации объекта или определения каких-либо его свойств по его изображению (оптическое распознавание) или аудиозаписи (акустическое распознавание) и другим характеристикам.

Образ — классификационная группировка в системе классификации, объединяющая (выделяющая) определенную группу объектов по некоторому признаку. Образы обладают характерным свойством, проявляющимся в том, что ознакомление с конечным числом явлений

из одного и того же множества дает возможность узнавать сколь угодно большое число его представителей. Образы обладают характерными объективными свойствами в том смысле, что разные люди, обучающиеся на различном материале наблюдений, большей частью одинаково и независимо друг от друга классифицируют одни и те же объекты. В классической постановке задачи распознавания универсальное множество разбивается на части-образы. Каждое отображение какого-либо объекта на воспринимающие органы распознающей системы, независимо от его положения относительно этих органов, принято называть изображением объекта, а множества таких изображений, объединенные какими-либо общими свойствами, представляют собой образы.

В целом можно выделить три метода распознавания образов.

Первый метод — *метод перебора*. В этом случае производится сравнение с базой данных, где для каждого вида объектов представлены всевозможные модификации отображения. Например, для оптического распознавания образов можно применить метод перебора вида объекта под различными углами, масштабами, смещениями, деформациями и т. д. Для букв нужно перебирать шрифт, свойства шрифта и т. д. В случае распознавания звуковых образов, соответственно, происходит сравнение с некоторыми известными шаблонами (например, слово, произнесенное несколькими людьми).

Второй метод связан с более глубоким анализом характеристик образа. В случае оптического распознавания это может быть определение различных геометрических характеристик. Звуковой образец в этом случае подвергается частотному, амплитудному анализу и т. д.

Третий метод — это использование нейронных сетей. Этот метод требует либо большого количества примеров задачи распознавания при обучении, либо специальной структуры нейронной сети, учитывающей специфику данной задачи. Тем не менее, его отличает более высокая эффективность и производительность.

Задачи распознавания имеют следующие характерные черты.

- преобразование исходных данных к виду, удобному для распознавания;
- собственно распознавание (указание принадлежности объекта определенному классу).

В этих задачах можно вводить понятие аналогии или подобия объектов и формулировать правила, на основании которых объект зачисляется в один и тот же класс или в разные классы.

Также можно оперировать набором прецедентов-примеров, классификация которых известна и которые в виде формализованных описаний могут быть предъявлены алгоритму распознавания для настройки на задачу в процессе обучения.

Для этих задач трудно строить формальные теории и применять классические математические методы (часто недоступна информация для точной математической модели или выигрыш от использования модели и математических методов несоизмерим с затратами).

Типы задач распознавания:

- задача распознавания или отнесение предъявленного объекта по его описанию к одному из заданных классов (обучение с учителем);
- задача автоматической классификации или разбиение множества объектов, ситуаций, явлений по их описаниям на систему непересекающихся классов (таксономия, кластерный анализ, самообучение);
- задача выбора информативного набора признаков при распознавании;
- задача приведения исходных данных к виду, удобному для распознавания;
- динамическое распознавание и динамическая классификация для динамических объектов;
- задача прогнозирования или вариант предыдущей задачи, в котором решение должно относиться к некоторому моменту в будущем.

Сети свертки. Заметим, что при использовании полносвязной нейронной сети прямого распространения для обработки изображений приходится выражать двумерную матрицу изображения в виде одномерного вектора, и тогда получается, что когда мы преобразуем изображение в линейную цепочку байт, мы что-то безвозвратно теряем. Причем с каждым слоем эта потеря только усугубляется — мы теряем топологию изображения, т. е. взаимосвязь между отдельными его частями. Задача распознавания подразумевает умение нейросети быть устойчивой к небольшим сдвигам, поворотам и изменению масштаба изображения, т. е. она должна извлекать из данных некие инварианты, не зависящие от почерка того или иного человека.

Решение этой проблемы было найдено американским ученым французского происхождения Яном ЛеКуном, вдохновленным рабо-

тами нобелевских лауреатов в области медицины Torsten Nils Wiesel и David H. Hubel. Эти ученые исследовали зрительную кору головного мозга кошки и обнаружили, что существуют так называемые простые клетки, которые особенно сильно реагируют на прямые линии под разными углами, и сложные клетки, которые реагируют на движение линий в одном направлении. Ян ЛеКун предложил использовать так называемые сверточные нейронные сети.

Сети свертки представляют собой многослойный персептрон, специально созданный для распознавания двумерных поверхностей с высокой степенью инвариантности к преобразованиям, масштабированию, искажениям и прочим видам деформации.

Идея сверточных нейронных сетей заключается в чередовании сверточных слоев (*C-layers*), субдискретизирующих (subsampling) слоев (*S-layers*) и наличии полносвязных (*F-layers*) слоев на выходе (рис. 5.27).

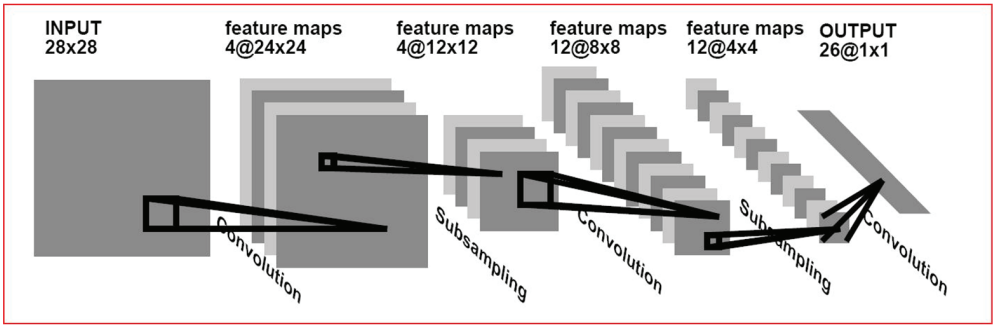


Рис. 5.27. Архитектура сверточной нейронной сети

Такая архитектура заключает в себе три основные парадигмы:

- Локальное восприятие.
- Разделяемые веса.
- Субдискретизация (локальное усреднение).

Локальное восприятие подразумевает, что на вход одного нейрона подается не все изображение (или выходы предыдущего слоя), а лишь некоторая его область. Такой подход позволил сохранять топологию изображения от слоя к слою.

Концепция *разделяемых весов* предполагает, что для большого количества связей используется очень небольшой набор весов. Например, если имеется на входе изображение размерами 28×28 пикселей, то каждый из нейронов следующего слоя примет на вход только небольшой

участок этого изображения размером, к примеру, 5×5, причем каждый из фрагментов будет обработан одним и тем же набором. Важно понимать, что самих наборов весов может быть много, но каждый из них будет применен ко **всему** изображению. Такие наборы часто называют ядрами (*kernels*). Нетрудно посчитать, что даже для 10 ядер размером 5×5 для входного изображения размерами 28×28 количество связей окажется равным примерно 256000, а количество настраиваемых параметров всего 250, что значительно меньше, чем у полносвязанной сети прямого распространения.

Искусственно введенное ограничение на веса улучшает обобщающие свойства сети, что в итоге позитивно сказывается на способности сети находить инварианты в изображении и реагировать главным образом на них, не обращая внимания на прочий шум. Большинство систем распознавания образов строятся на основе двумерных фильтров. Фильтр представляет собой матрицу коэффициентов, обычно заданную вручную. Эта матрица применяется к изображению с помощью математической операции, называемой *сверткой*. Суть этой операции в том, что каждый фрагмент изображения умножается на матрицу (ядро) свертки поэлементно и результат суммируется и записывается в аналогичную позицию выходного изображения. Основное свойство таких фильтров заключается в том, что значение их выхода тем больше, чем больше фрагмент изображения похож на сам фильтр. Таким образом, изображение, свернутое с неким ядром, даст нам другое изображение, каждый пиксель которого будет означать степень похожести фрагмента изображения на фильтр. Иными словами это будет карта признаков.

К основным особенностям сетей свертки стоит отнести четыре процедуры.

Извлечение признаков. Каждый нейрон получает входной сигнал от локального рецептивного поля в предыдущем слое, извлекая таким образом его локальные признаки. Как только признак извлечен, его точное местоположение не имеет значения, поскольку приблизительно установлено его расположение относительно других признаков.

Отображение признаков. Каждый вычислительный слой сети состоит из множества карт признаков, каждая из которых имеет форму плоскости, на которой все нейроны должны совместно использовать одно и то же множество синаптических весов. Эта форма структурных ограничений имеет следующие преимущества.

Инвариантность к смещению, реализованную посредством карт признаков с использованием свертки с ядром небольшого размера, выполняющим функцию «сплющивания».

Сокращение числа свободных параметров, реализованное с помощью совместного использования синаптических весов.

За каждым слоем свертки следует вычислительный слой, осуществляющий локальное усреднение. Посредством этого достигается уменьшение разрешения для карт признаков. Эта операция приводит к уменьшению чувствительности выходного сигнала оператора отображения признаков к смещению и прочим формам деформации.

Процесс распространения сигнала в *C*-слое изображен на рис. 5.28.

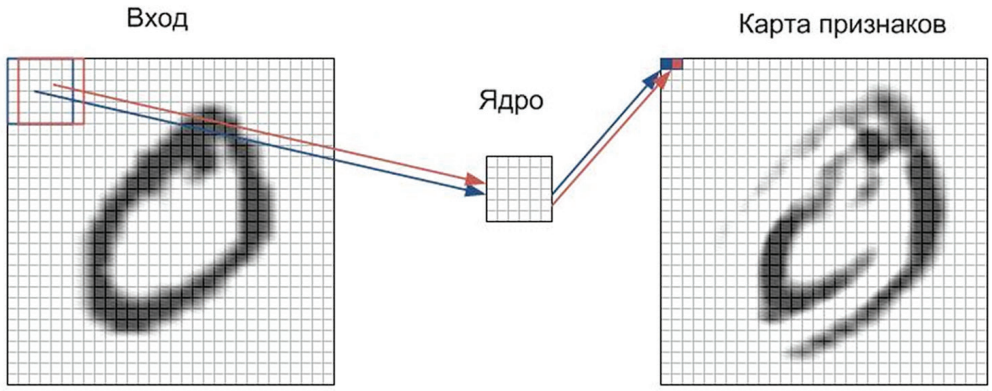


Рис. 5.28. Процесс распространения сигнала в сверточном слое

Каждый фрагмент изображения поэлементно умножается на небольшую матрицу весов (ядро), результат суммируется. Эта сумма является пикселем выходного изображения, которое называется *картой признаков*. Здесь мы опустили тот факт, что взвешенная сумма входов еще пропускается через функцию активации (как в любой другой нейросети). На самом деле это может происходить и в *S*-слое, принципиальной разницы тут нет. Следует сказать, что в идеале не разные фрагменты проходят последовательно через ядро, а параллельно все изображение проходит через идентичные ядра. Кроме того, количество ядер (наборов весов) определяется разработчиком и зависит от того какое количество признаков необходимо выделить. Еще одна особенность сверточного слоя в том, что он немного уменьшает изображение за счет краевых эффектов.

Суть *субдискретизации* и *S*-слоев заключается в уменьшении пространственной размерности изображения, входное изображение грубо (усреднением) уменьшается в заданное количество раз. Чаще всего в 2 раза, хотя может быть и неравномерное изменение, например, 2 по вертикали и 3 по горизонтали. Субдискретизация нужна для обеспечения инвариантности к масштабу.

Чередование слоев позволяет составлять карты признаков из карт признаков, что на практике означает способность распознавания сложных иерархий признаков.

Обычно после прохождения нескольких слоев карта признаков вырождается в вектор или даже скаляр, но таких карт признаков становится сотни. В таком виде они подаются на один-два слоя полносвязной сети. Выходной слой такой сети может иметь различные функции активации. В простейшем случае это может быть тангенциальная функция.

На рис. 5.27 показана архитектурная схема сверточной сети, состоящей из одного входного, четырех скрытых и одного выходного слоя нейронов. Эта сеть была создана для обработки изображений, в частности, при распознавании рукописного текста. Входной слой, состоящий из матрицы 28×28 сенсорных узлов, получает изображения различных символов, которые предварительно смещены к центру и нормализованы по размеру. После этого вычислительные слои поочередно реализуют операции свертки и усреднения, как описывается ниже.

Первый скрытый слой выполняет свертку. Он состоит из четырех карт признаков, каждая из которых представляет собой матрицу из 24×24 нейронов. Каждому нейрону соответствует поле чувствительности размером 5×5.

Второй скрытый слой выполняет локальное усреднение. Он тоже состоит из четырех карт признаков, но теперь уже содержащий матрицы размером 12×12 нейронов. Каждому нейрону соответствует рецептивное поле размером 2×2, настраиваемый коэффициент, настраиваемый порог и сигмоидальная функция активации. Настраиваемый коэффициент и порог определяют рабочую область нейрона.

Третий скрытый слой выполняет повторную свертку. Он состоит из 12 карт признаков, каждая из которых представляет собой матрицу из 8×8 нейронов. Каждый нейрон этого скрытого слоя имеет синаптические связки с различными картами признаков предыдущего скрытого слоя.

Четвертый скрытый слой осуществляет повторное локальное усреднение и состоит из 12 карт признаков, каждая из которых — матрица 4×4.

Выходной слой осуществляет последний этап свертки.

При последовательном прохождении сверточной сети количество карт признаков увеличивается при одновременном уменьшении пространственного разрешения. Многослойный персептрон содержит приблизительно 100 000 синаптических связей, и в то же время имеет только 2600 свободных параметров. Такое сокращение количества свободных параметров было получено за счет совместного использования весов.

Использоваться уже обученная сверточная нейронная сеть должна на подобных, можно сказать, аналогичных изображениях, потому что любое даже не очень существенное изменение базовых характеристик изображений потребует переобучения сети. Однако обучить подобную нейронную сеть для распознавания образов, в данном случае самого простого примера — распознаванию рукописных цифр, нужны, как минимум, десятки часов на современном аппаратном обеспечении.

Пример 5.2. Нейронная сеть для распознавания образов — рукописных цифр

Создадим сверточную нейронную сеть для распознавания рукописных цифр от 0 до 9. Для ее обучения используем обучающую базу с домашней страницы профессора Яна Ле-Куна (http://yann.lecun.com/exdb/mnist/index.html). Название базы данных MNIST. Она содержит 60 000 обучающих пар (изображение — метка) и 10 000 тестовых (изображения без меток). Изображения нормализованы по размеру и отцентрированы. Размер каждой цифры не более 20×20 элементов, они вписаны в квадрат размером 28×28 элементов. Пример первых 12 цифр из обучающего набора базы MNIST приведен на рис. 5.29.

Рис. 5.29. 12 первых цифр из обучающего набора базы MNIST

База отлично подходит для тех, кому необходимо опробовать методы обучения на реальных данных без дополнительных затрат усилий на форматирование и предобработку.

База MNIST состоит из четырех файлов: двух файлов для обучения нейронной сети train-images.idx3-ubyte и train-labels.idx1-ubyte и двух файлов для тестирования обученной нейронной сети t10k-images.idx3-ubyte и t10k-labels.idx1-ubyte.

Суть распознавания рукописного текста сводится к тому, что нет абсолютно одинаковых почерков, каждый человек пишет цифры поразному, и методом точного перебора и сравнения тут не обойтись. Именно поэтому и используются свойства нейронных сетей, так как хорошо обученная нейронная сеть может с высокой долей вероятности определять, что за цифра будет написана — даже если такая цифра немного отличается от тех, какие были в обучающей базе данных. Для валидации нейронной сети следует подготовить образцы изображений цифр, написанных собственным почерком.

В ходе работы с сетью можно проверить, насколько хорошо обучена нейронная сеть, а также наглядно посмотреть на цифры в обучающей выборке, сравнить их со своим подчерком, увидеть величины возбуждений нейронов на выходе нейронной сети. Все действия с нейронной сетью проводятся в специально разработанном графическом интерфейсе, использующем *m*-функции приложения nnet системы моделирования MATLAB.

Результат работы интерфейса представлен на рис. 5.30, на котором можно просмотреть цифры из обучающей выборки — файла train-images.idx3-ubyte. Помимо этого, при запуске приложения идет загрузка нейронной сети и если нейронная сеть обучена, то при нажатии на кнопку Recognize данные с поля Icon edit pane пойдут на вход нейронной сети. После обработки данных результат работы — информация на выходе нейронной сети — будет отображена в окне Recognition Result. На рисунке видно изображение третьей цифры из обучающей выборки. Такими цифрами обучается нейронная сеть.

Обновив поле, можно рисовать собственные цифры и посылать информацию на вход сверточной нейронной сети. В зависимости от того, какой из нейронов на выходе возбудился больше всех, такая цифра и будет распознана нейронной сетью. В рабочее пространство системы MATLAB выводятся все значения возбуждения всех десяти выходных нейронов. По этим числам можно сделать вывод о том, что уровень возбуждения нейронов фактически является вычисленной вероятностью того, какая цифра подается на вход нейронной сети. Таким образом, после каждого запуска этой нейронной сети создается вектор из деся-

ти чисел, максимальное из которых с определенной долей вероятности распознает рукописную цифру на предъявленном изображении.

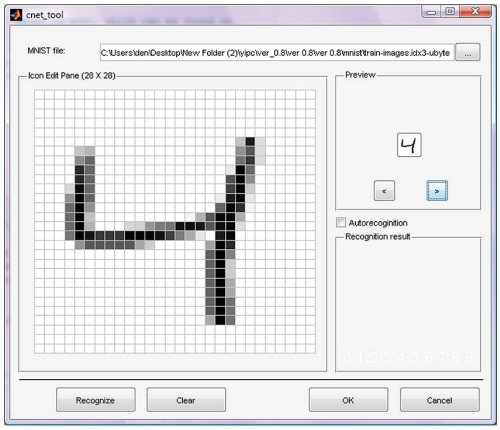


Рис. 5.30. Графический интерфейс для проверки работы нейронной сети

После проведения ряда запусков составляется таблица условных вероятностей (табл. 5.2), анализ которой позволит дать ответ на вопрос о качестве распознавания рукописных цифр, созданных нейронной сетью. Очевидно, такой статистический анализ может быть поддержан мощным статистическим инструментарием системы MATLAB.

Таблица 5.2

Результаты распознавания цифры 4

Цифры для распознавания	Значения возбуждения нейронов на выходе нейронной сети		
	Первый опыт	Второй опыт	Третий опыт
0	–1,1068	–1,1177	–1,0448
1	–0,8403	–0,8939	–0,7874
2	–0,9998	–1,0148	–1,0816
3	–1,0266	–0,9504	–1,1228
4	**0,8786**	**0,9338**	**0,8414**
5	–1,0510	–0,9911	–1,1078
6	–1,0371	–1,0157	–1,0744
7	–1,1835	–1,1521	–1,1206
8	–1,0083	–0,9128	–1,0599
9	–0,9623	–0,8717	–0,9918

Заканчивая главу о нейронных сетях, отметим, что нейронные сети — это раздел искусственного интеллекта, в котором для обработки сигналов используются явления, аналогичные происходящим в нейронах живых существ. Современное аппаратное обеспечение пока только приближается к возможностям и огромному потенциалу в параллельной обработке информации человеческим мозгом. Поэтому принципиальное противоречие между программной и аппаратной составляющими нейронных сетей служит фундаментальным двигателем прогресса в этой области и дает надежду на прорывные достижения.

Вопросы и задания к главе 5

1. Назовите основные приложения нейронных сетей. Какие области искусственного интеллекта представляют нейронные сети?
2. В чем состоят достоинства и недостатки нейронных сетей?
3. Как выглядит и «работает» модель персептрона?
4. Как влияет на качество обработки входных сигналов в персептроне их количество?
5. Что такое функция возбуждения и почему в модели нейрона она обычно имеет пороговый вид?
6. В чем состоит отличие нейрона от персептрона?
7. Дайте определение многослойного персептрона. Какие слои этого персептрона являются скрытыми?
8. Почему функции возбуждения многослойного персептрона обязательно должны быть нелинейными?
9. Укажите последовательность шагов использования многослойного персептрона при решении информационных задач.
10. Можно ли считать нейронную сеть в системе MATLAB многослойным персептроном?
11. Что означает условие «персептрон прямого распространения» и каким образом это условие применяется к нейронным сетям в системе MATLAB?
12. Повторите действия, описанные в примере 5.2, и получите вероятностные характеристики для другой рукописной цифры.

Библиографический список

1. Кузин Л. Т. Основы кибернетики. Т. 1 : Математические основы кибернетики / Л. Т. Кузин. Москва : Энергоатомиздат, 1994. 576 с.
2. Кормен Т. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. Москва : МЦНМО, 2001. 960 с.
3. Форд Л. Потоки в сетях / Л. Форд, Д. Фалкерсон. Москва : Мир, 1966. 277 с.
4. Схрейвер А. Теория линейного и целочисленного программирования / А. Схрейвер. Москва : Мир, 1991. Т. 1, 2.
5. Ануфриев И. Е. MATLAB 7 / И. Е. Ануфриев, А. Б. Смирнов, Е. Н. Смирнова. Санкт-Петербург : БХВ-Петербург, 2005. 1104 с.
6. Zhang Y. Solving Large-Scale Linear Programs by Interior-Point Methods Under the MATLAB Environment / Y. Zhang // Technical Report TR96–01 ; Department of Mathematics and Statistics University of Maryland. Baltimore, MD, July, 1995.
7. Mehrotra S. On the Implementation of a Primal-Dual Interior Point Method / S. Mehrotra // SIAM Journal on Optimization. 1992. Vol. 2, pp. 575–601.
8. Kuhn H. W. The Hungarian Method for the Assignment Problem / H. W. Kuhn // Naval Research Logistics Quarterly. 1955. Т. 2. pp. 83–97.
9. Burkard R. E. Assignment Problems / R. E. Burkard, M. Dell'Amico, S. Martello // SIAM, Philadelphia (PA.), 2009.
10. [Электронный ресурс] Режим доступа: http://algolist.manual.ru/maths/graphs/maxflows/Goldberg_Rao.php. Загл. с экрана.
11. Советов Б. Я. Моделирование систем: учебник для вузов / Б. Я. Советов, С. А. Яковлев. Москва : Высшая школа, 1998. 319 с.
12. Питерсон Дж. Теория сетей Петри и моделирование систем / Дж. Питерсон. Москва : Мир, 1984. 264 с.

13. Котов В. Е. Сети Петри / В. Е. Котов. Москва : Наука, 1984. 160 с.
14. Ломазова И. А. Вложенные сети Петри: Моделирование и анализ распределенных систем с объектной структурой / И. А. Ломазова. Москва : Научный мир, 2004. 208 с.
15. Доррер Г. А. Технология моделирования и разработки учебных электронных изданий / Г. А. Доррер, Г. М. Рудакова ; под ред. В. С. Соколова. Новосибирск : Изд-во СО РАН, 2006. 272 с.
16. Доррер А. Г. Моделирование процесса обучения с помощью вложенных сетей Петри / А. Г. Доррер, Г. А. Доррер, Г. М. Рудакова. Россия, 2006.
17. Зайцев Д. А. Моделирование телекоммуникационных систем в CPN Tools : учебное пособие по курсу «Математическое моделирование информационных систем» / Д. А. Зайцев, Т. Р. Шмелева. Одесса : Одесская национальная академия связи им. А. С. Попова, 2006. 60 с.
18. Jensen K. Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. V.1 : Basic Concepts / K. Jensen. Springer-Verlag, 1992.
19. Jensen K. Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. V. 2: Analysis Methods / K. Jensen. Springer-Verlag, 1994.
20. Jensen K. Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3: Practical Use / K. Jensen. Springer-Verlag, 1997.
21. Wang J. Timed Petri Nets, Theory and Application / J. Wang. Kluwer Academic Publishers, 1998.
22. Billington J. Application of Petri Nets to Communication Networks ; editors M. Diaz and G. Rozenberg. V. 1605 / J. Billington. Springer-Verlag, 1999.
23. Зыбарев Ю. М. Спецификация функциональной модели информационного портала сетями Петри / Ю. М. Зыбарев, С. П. Чернев. (ser@sibinfo.ru). [Электронный ресурс] Режим доступа: http://zhurnal.ape.relarn.ru/articles/2003/095.pdf 1057. Загл. с экрана.
24. Svadova Martina. PN Matlab Toolbox 2.0/Martina Svadova, Zdenek Hanzalek // Center for Applied Cybernetics, Czech Technical University. Prague, 2001. [Электронный ресурс] Режим доступа: https://www2.humusoft.cz/www/papers/tcp04/svadova.pdf

25. Трухин М. П. Моделирование сигналов и систем : учеб. пособие. Ч. 1 / М. П. Трухин. Екатеринбург : УГТУ-УПИ, 2006. 210 с.
26. Мурата Т. Сети Петри: свойства, анализ, приложения / Т. Мурата // Труды ТИИЭР. 1989. Т. 77, № 4. С. 41–85.
27. Цапко С. Г. Объектное представление имитационных моделей подсистем сложной технической системы в терминах *E*-сети // С. Г. Цапко, И. В. Цапко. Томск : Известия Томского политехнического университета, 2006. Т. 309, № 5. С. 167–170. [Электронный ресурс] Режим доступа: http://www.lib.tpu.ru/fulltext/v/Bulletin_TPU/2006/v309/i5/35.pdf. Загл. с экрана.
28. Ефремов А. А. *E*-сетевое моделирование надежности последовательно-параллельных систем с восстановлением / А. А. Ефремов. Томск : Известия Томского политехнического университета, 2006. Т. 309, № 7. С. 97–101. [Электронный ресурс] Режим доступа: http://www.lib.tpu.ru/fulltext/v/Bulletin_TPU/2006/v309/i7/21.pdf. Загл. с экрана.
29. Дмитриева Е. А. Система *E*-сетевого имитационного моделирования EVA / Е. А. Дмитриева. Томск : Изд-во ТПУ, 1994. 31 с.
30. Губин И. Г. Разработка систем автоматизированного проектирования (САПР) : учебно-методическое пособие / И. Г. Губин. Томск : Томский межвузовский центр дистанционного образования, 2001. 38 с. [Электронный ресурс] Режим доступа: http://www.st10.reshaem.net/tasks/task_139648.pdf. Загл. с экрана.
31. Банников А. А. Представление *E*-сетей с помощью сетей Петри для обеспечения возможности матричного анализа / А. А. Банников, Д. В. Фатхи. Ростов-на-Дону : Молодой исследователь Дона, 2016. Т. 1. С. 1–7. [Электронный ресурс] Режим доступа: http://mid-journal.ru. Загл. с экрана.
32. Кнут Д. Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск ; пер. с англ. / Д. Кнут. Москва: Мир, 1978. 844 с.
33. Кормен Т. Алгоритмы: построение и анализ ; пер. с англ. / Т. Кормен, Ч. Лейзерсон, Р. Ривест. Москва : МЦМНО, 2001. 960 с.
34. Haykin S. Adaptive Filter Theory; 4th Ed./S. Haykin. Prentice Hall, 2002. 936 p.
35. Widrow B. Adaptive Noise Cancelling: Principles and Applications / B. Widrow, J. R. Glover, J. M. McCool // Proceedings of the IEEE, 1975. Vol. 63, No. 12, pp. 1692–1716.

36. Widrow B. Adaptive switching circuits / B. Widrow, M. Hoff // Proc. IRE WESCON Convention Record, 1960. Pp. 107–115.
37. Хайкин С. Нейронные сети: полный курс. 2-е изд. / С. Хайкин ; пер. с англ. под ред. Н. Н. Куссуль. Москва : Издательский дом «Вильямс», 2006. 1104 с.
38. Основский С. Нейронные сети для обработки информации / С. Основский ; пер. с польск. И. Д. Рудинского. Москва : Финансы и статистика, 2004. 344 с.
39. LeCun Y. Convolutional Networks for Images, Speech, and Time-Series. The Handbook of Brain Theory and Neural Networks / Y. LeCun, Y. Bengio. MIT Press, 1995.
40. Медведев В. С. Нейронные сети. MATLAB 6 / В. С. Медведев, В. Г. Потемкин ; под общ. ред. В. Г. Потемкина. Москва : ДИАЛОГ-МИФИ, 2002. 496 с.

Учебное издание

Трухин Михаил Павлович

МОДЕЛИРОВАНИЕ СИГНАЛОВ И СИСТЕМ

Сетевые модели

Редактор Н. П. Кубыщенко
Верстка О. П. Игнатьевой

Подписано в печать 23.11.2018. Формат 70×100/16.
Бумага офсетная. Цифровая печать. Усл. печ. л. 16,4.
Уч.-изд. л. 8,6. Тираж 40 экз. Заказ 282

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: +7 (343) 375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620083, Екатеринбург, ул. Тургенева, 4
Тел.: +7 (343) 358-93-06, 350-58-20, 350-90-13
Факс: +7 (343) 358-93-06
http://print.urfu.ru

9 785799 625030